IIIA Institut d'Investigació en Intel·ligència Artificial

CSIC Consejo Superior de Investigaciones Científicas

New Solving Techniques for Maximum and Minimum Satisfiability

by

Juan Ramón SOLER CABREJAS

A dissertation presented in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy in Computer Science

Advisor:
Felip Manyà

Tutor:
Eva Armengol

November 15, 2020

UAB
Universitat Autònoma de Barcelona
Departament de Ciències de la Computació
Programa de Doctorat en Informàtica

Abstract

The Satisfiability problem, or SAT, is the problem of deciding if there exists an assignment that satisfies a given propositional formula. SAT was the first problem proven to be NP-complete and is one of the most studied problems in Computer Science. On the other hand, MaxSAT and MinSAT are optimization versions of SAT where the goal is to find an assignment that maximizes or minimizes the number of satisfied clauses, respectively. All these problems are significant because many practical problems can be encoded as SAT, MaxSAT or MinSAT problems, and solved using a SAT, MaxSAT or MinSAT solver. While SAT is used to solve decision problems, MaxSAT and MinSAT are used to solve optimization problems.

The general objective of this PhD thesis is to advance the state of the art in solving computationally difficult optimization problems by reducing them to MaxSAT and MinSAT. To achieve this objective, we have investigated new inference systems for MaxSAT and MinSAT based on semantic tableaux, and suitable encodings for new MaxSAT applications.

Regarding inference systems, the thesis defines a complete tableau calculus for solving clausal MaxSAT, a complete tableau calculus for solving clausal MinSAT and a complete tableau calculus for solving both clausal MaxSAT and clausal MinSAT. Moreover, the thesis proposes two different approaches to solving non-clausal MaxSAT and non-clausal MinSAT: in the first approach, the thesis defines novel cost-preserving transformations from non-clausal MaxSAT to clausal MaxSAT. Such transformations are then extended to define cost-preserving transformations from non-clausal MinSAT to clausal MinSAT. In the second approach, the thesis defines a genuine tableau calculus for non-clausal MaxSAT and proves its soundness and completeness. It also describes how the tableau calculus for non-clausal MaxSAT can be used to solve non-clausal MinSAT.

Regarding new MaxSAT applications, the thesis defines and empirically evaluates MaxSAT encodings for the team composition problem in a classroom and proves that this problem is NP-hard. The insights gained are useful for solving other challenging optimization problems via their reduction to MaxSAT. In particular, the thesis shows how to solve more complex team formation problems, using the synergistic team composition model as a case study.

Acknowledgments

None of this would have been possible without the help of many people. All of them have my gratitude.

I want to express my deep gratitude to my advisor, Felip Manyà. His long time support, encouragement and guidance have been invaluable for me to develop this thesis and to orient my career. Thank you for being my pole star in this adventure, you always will have my admiration and affection.

I also want to express my deep gratitude to my thesis committee members. Thanks to Carlos Ansótegui, Jordi Levy and Mateu Villaret for all their helpful suggestions and time.

Many thanks to the people at the IIIA. The people who I collaborated with at some point: Lluís Godo and Tommaso Flaminio. The people who advised or gave me hints or ideas: Pere Garcia, Pedro Meseguer, Pilar Dellunde, Juan Antonio Rodríguez and my tutor, Eva Armengol. The administrative staff: many thanks Ana, Montse and Eva for your patience with my clumsiness. And especially to the other PhD students of the IIIA. The students who I spent the most time with: David, Toni, Adán and Oguz. The students who I could only spend a not so long time with: Albert, Sergi, Ewa, Kemo, Paula and Núria. Thanks also to my coauthors Josep Argelich, Santiago Negrete, Carme Roig, and specially Chu Min Li, with whom I hope to meet in person one day soon.

Last, but most important, I want to thank my family. To my partner Tania, for your patience and emotional support in this journey. To my mother Mercedes and my father Ramon, for your inexhaustible patience and unconditional support for so many years. To my brother Daniel, for always reminding me of the importance of this work. This thesis is dedicated to all of you. Thanks to your help, I believed in my possibilities.

This research was partially funded by the MINECO FPI grant BES-2016-078447, the MINECO-FEDER project RASO (TIN2015-71799-C2-1-P/2-P), the MICINN project ISINC (PID2019-111544GB-C21), and the Project LOGISTAR from the EU H2020 Research and Innovation Programme under Grant Agreement No. 769142.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The Satisfiability problem, or SAT, is the problem of deciding if there exists an assignment that satisfies a given propositional formula. SAT is one of the most studied problems in Computer Science, becoming central both in its practical and theoretical aspects.

According to the theory of computational complexity, SAT was the first problem proven to be NP-complete by Cook [71] and Levin [127]. A problem is said to be NP-complete when any NP problem can be reduced to it and, unless P=NP, there is no algorithm to solve it in polynomial time.

One important use of SAT has been theoretical. By knowing that SAT is NP-complete, we can prove the NP-completeness of other problems. Since any NP problem can be reduced to SAT, reducing SAT to a given new problem P demonstrates the NP-completeness of P.

The irruption of novel SAT solving techniques has extended the importance of SAT to the practical field. Although SAT has an exponential time complexity in the worst case, modern SAT solvers have shown that many challenging real-world problems encoded as SAT instances can be solved in a reasonable amount of time. Because of this, SAT is currently a powerful generic approach to solving decision problems in relevant areas as automatic circuit design [67, 167], formal verification of hardware and software [56, 57, 69, 70, 81, 115, 189, 190], scheduling [51, 89, 200], planning [117, 118, 181], bioinformatics [151, 152], cryptography [83], manufacturing [82] and mathematical problems [98, 99, 120, 199].

On the other hand, MaxSAT is an optimization version of SAT whose objective is to find an assignment that maximizes (minimizes) the number of satisfied (unsatisfied) clauses in a given multiset of clauses. Another optimization version of SAT is MinSAT; in this case, the objective is to minimize (maximize) the number of satisfied (unsatisfied) clauses. Both optimization problems belong to the NP-hard complexity class. MaxSAT and MinSAT are also complete for the class FP^{NP} [175], which includes the set of function problems computable in polynomial time using an NP oracle. The FP^{NP} class includes many practical optimization problems and, by completeness, all of them can be compactly encoded to MaxSAT or MinSAT [78, 98].

We use the terms MaxSAT and MinSAT in a broad sense: we allow to distinguish between hard and soft clauses, and associate a weight to each soft clause that indicates the penalty of its violation. These more general formulations of MaxSAT and MinSAT are technically known as Weighted Partial MaxSAT and Weighted Partial MinSAT, respectively. An optimal Weighted Partial MaxSAT (MinSAT) solution is a truth assignment that satisfies all the hard clauses and minimizes (maximizes) the sum of weights of violated soft clauses.

Nowadays, MaxSAT and MinSAT are very competitive generic problem solving approaches to solving combinatorial optimization problems [23, 148, 169]. Applications of MaxSAT and MinSAT have also increased in number, and include real-world domains as diverse as bioinformatics [68, 90, 160], combinatorial testing [30, 197], combinatorial auctions [148], circuit design and debugging [182], community detection in complex networks [111], diagnosis [74], FPGA routing [196], planning [202], scheduling [60], team formation [158] and time tabling [9], among many others.

MaxSAT is much more developed than MinSAT, as witnessed by the annual MaxSAT Evaluation [35, 47, 84] and the number of publications in top Artificial Intelligence journals and conferences. Especially in the case of MaxSAT, the scientific community has made a remarkable effort to implement exact solvers, achieving dramatic improvements in performance. Roughly speaking, we find three main groups of MaxSAT solvers: branch-and-bound solvers, SAT-based solvers and solvers based on the Implicit Hitting Set (IHS) approach. Although in this thesis we mostly concentrate on complete solvers, there are also incomplete approaches for solving MaxSAT and MinSAT [126, 171].

Branch-and-bound MaxSAT solvers implement the branch-and-bound scheme and incorporate good-quality lower bounds that detect inconsistent subsets of clauses by applying unit propagation. They also apply some inference rules at each node of the search tree. Representative solvers of this group are MaxSatz [135, 138], MiniMaxSat [94], Ahmaxsat [2] and Akmaxsat [122].

SAT-based MaxSAT algorithms proceed by reformulating the MaxSAT optimization problem into a sequence of SAT decision problems. Each SAT instance of the sequence encodes whether there exists an assignment to the MaxSAT instance with a cost less than or equal to a certain k. SAT instances with a k less than the optimal cost are unsatisfiable, while the others are satisfiable. The SAT solver is executed in incremental mode in order to keep the clauses learnt at each iteration over the sequence of SAT instances. There are two main types of SAT-based MaxSAT solvers: model-guided and core-guided. Model-guided solvers iteratively refine (decrease) the upper bound and guide the search with satisfying assignments obtained from satisfiable SAT instances. Core-guided solvers iteratively refine (increase) the lower bound and guide the search with the unsatisfiable cores obtained from unsatisfiable SAT instances. Both have strengths and weaknesses, also existing hybrid approaches. Representative solvers of this group are msu1.2 [161, 162], WBO [153, 154], Open-WBO [166], WPM1 [21], PM2 [23], WPM2 [22], WPM3 [29], Eva [172], SAT4J-Maxsat [54, 55], QMaxSat [121, 198] and Pacose [84].

IHS-based MaxSAT solvers alternate two phases, the optimization and the propositional reasoning phases. The optimization reasoning is carried out by a minimum cost hitting set solver, usually a MIP solver, which is good for arithmetic reasoning. On the other hand, the SAT solver is only used for propositional reasoning, checking the satisfiability of subsets of the original problem which are returned by the MIP solver. This separation allows specialization. The most representative solvers of this group are LMHS [183] and MaxHS [46, 76, 77, 78, 79, 100].

In the case of MinSAT, there exist an implementation of a branch-and-bound MinSAT solver [148] and two SAT-based MinSAT solvers [97, 148].

1.1 Motivation and objectives

The success of MaxSAT and MinSAT is due to both the definition of suitable encodings and the incorporation of powerful solving techniques in modern MaxSAT and MinSAT solvers. Given the relevance of encodings and solving techniques, the general objective of this thesis is to advance the state of the art of MaxSAT and MinSAT-based problem solving by defining new encodings for challenging optimization problems and devising new solving methods based on logic.

Regarding encodings, our objective is to investigate how to efficiently solve the problem of creating teams in a classroom by defining suitable MaxSAT encodings and then solving the resulting encodings with a modern MaxSAT solver. This problem had not been solved using MaxSAT technology until we started the PhD thesis, and we selected it because it is a computationally difficult problem with interesting applications in the domain of education. We focused on MaxSAT because there are several extremely competitive MaxSAT solvers, but the contributed encodings can be easily adapted to MinSAT.

Regarding logic-based methods for MaxSAT and MinSAT, our objective is to investigate how tableau-based approaches can be extended to both MaxSAT and MinSAT. Tableau calculi for MaxSAT and MinSAT had not been defined until we started the PhD thesis, and the only logical approach defined for MaxSAT and MinSAT was based on resolution [64, 132].

As in SAT, resolution-based methods are probably more powerful than tableau-based methods for solving clausal MaxSAT and MinSAT, but present some limitations when dealing with more expressive formalisms. The advantage of tableaux over resolution is that tableaux allow one to solve directly non-clausal MaxSAT and MinSAT, can be naturally extended to first-order logic and are more suitable for dealing with non-classical logics in which it is not easy to define clausal forms.

Since we are interested in developing, for the first time in the SAT community, solving methods for non-clausal MaxSAT and MinSAT, another objective of this PhD thesis is to investigate clausal form transformations for arbitrary propositional MaxSAT and MinSAT instances.

It is important to highlight that the inference systems defined for SAT cannot be applied to MaxSAT and MinSAT. SAT inference rules preserve satisfiability but do not preserve the minimum/maximum number of unsatisfied clauses and, therefore, are unsound in MaxSAT and MinSAT. This fact has opened new research directions in computational logic and proof complexity[62], and the challenge now is to find out if the logical methods defined for MaxSAT and MinSAT can be used to improve SAT-based problem solving in the near future.

1.2 Contributions

The main original contributions of this PhD thesis are the following ones:

- The first contribution is the definition of complete tableau calculi for MaxSAT and MinSAT, as well as the definition of a complete calculus that is valid for both MaxSAT and MinSAT. Moreover, the thesis also describes how the mentioned calculi can be extended to deal with Weighted Partial MaxSAT and MinSAT instances. These calculi constitute the first approach to solving MaxSAT and MinSAT with semantic tableaux in the literature.
- The second contribution is the definition of three different cost-preserving clausal form transformations –called direct, improved and Tseitin-based transformations– from non-clausal MaxSAT to clausal MaxSAT. These transformations reduce non-clausal MaxSAT to clausal MaxSAT so that clausal MaxSAT solvers can be used to solve the MaxSAT problem of multisets of propositional formulas that are not necessarily in clausal form. The thesis also describes how the proposed transformations can be converted into cost-preserving transformations from non-clausal MinSAT to clausal MinSAT.
- The third contribution is the definition of a genuine complete tableau calculus for solving non-clausal MaxSAT. It is, to the best of our knowledge, the first inference system defined for non-clausal MaxSAT. Moreover, the tableau calculus for non-clausal MaxSAT can also be used to solve non-clausal MinSAT.
- The fourth contribution is the definition of the Team Composition Problem in a Classroom (TCPC) as a challenging optimization problem for MaxSAT solving, and the proof that TCPC is NP-hard. We define two MaxSAT encodings for TCPC –called maximizing and minimizing encodings– and provide empirical evidence that the minimizing encoding greatly outperforms the maximizing encoding. Using the synergistic team composition model as a case study, the thesis also shows how to reduce more complex team formation problems to MaxSAT.

1.3 Publications

Some of the results presented in this thesis have already been published in journals and conference proceedings. The list of publications can be divided into three groups: publications on clausal tableau calculi for MaxSAT and MinSAT, publications on non-clausal MaxSAT and publications on the team composition problem in a classroom.
The publications on clausal tableau calculi for MaxSAT and MinSAT are:

- Chu Min Li, Felip Manyà, Joan Ramon Soler: A Clause Tableau Calculus for MaxSAT. In Proceedings of **IJCAI 2016**, 766–772.
- Chu Min Li, Felip Manyà, Joan Ramon Soler: A Clause Tableau Calculus for MinSAT. In Proceedings of **CCIA 2016**, 88–97.
- Josep Argelich, Chu Min Li, Felip Manyà, Joan Ramon Soler: Clause Branching in MaxSAT and MinSAT. In Proceedings of **CCIA 2018**, 17–26.
- Josep Argelich, Chu Min Li, Felip Manyà, Joan Ramon Soler: Clause Tableaux for Maximum and Minimum Satisfiability. **Logic Journal of the IGPL 2020**.

The publications on non-clausal MaxSAT are:

- Chu-Min Li, Felip Manyà, Joan Ramon Soler: Clausal Form Transformation in MaxSAT. In Proceedings of **ISMVL 2019**, 132–137.
- Chu-Min Li, Felip Manyà, Joan Ramon Soler: A Tableau Calculus for Non-Clausal Maximum Satisfiability. In Proceedings of **TABLEAUX 2019**, 58–73.

The publications on the team composition problem in a classroom are:

- Felip Manyà, Santiago Negrete, Carme Roig, Joan Ramon Soler: A MaxSAT-based Approach to the Team Composition Problem in a Classroom. In: Autonomous Agents and Multiagent Systems - **AAMAS 2017 Workshops, Visionary Papers**. Springer LNCS 10643, 164–173.
- Felip Manyà, Santiago Negrete, Joan Ramon Soler: MaxSAT Instances of the Team Composition Problem in a Classroom. In Proceedings of **MaxSAT Evaluation 2018**.
- Felip Manyà, Santiago Negrete, Carme Roig, Joan Ramon Soler: Solving the Team Composition Problem in a Classroom. **Fundamenta Informaticae 2020** 174(1): 83–101.

1.4 Outline of the thesis

This section describes the structure of the document and presents an overview of the forthcoming chapters.

Chapter 2: Preliminaries

This chapter defines the MaxSAT and MinSAT problems and overviews the most relevant algorithms and techniques used to solve them. It first describes the main solving methods for exact MaxSAT: integer linear programming, MaxSAT resolution, branch-and-bound MaxSAT solvers, SAT-based MaxSAT solvers and implicit hitting set-based solvers. Then, it reviews some solving techniques for MinSAT and describes a branch-and-bound MinSAT solver. Moreover, it devotes a section to the MaxSAT Evaluation.

Chapter 3: Clausal Tableaux Calculi for MaxSAT and MinSAT

This chapter defines a tableau calculus for solving MaxSAT, a tableau calculus for solving MinSAT, and a tableau calculus for solving both MaxSAT and MinSAT. For each calculus, a proof of soundness and completeness is provided. It holds that the minimum number of contradictions derived among the branches of a completed MaxSAT tableau for a multiset of clauses ϕ is the minimum number of unsatisfied clauses in ϕ, and the maximum number of contradictions derived among the branches of a completed MinSAT tableau for a multiset of clauses ϕ' is the maximum number of unsatisfied clauses in ϕ'. It also describes how the new calculi can be extended to deal with Weighted Partial MaxSAT and MinSAT instances.

Chapter 4: Solving Non-Clausal MaxSAT and Non-Clausal MinSAT

This chapter proposes two approaches to solving non-clausal MaxSAT and non-clausal MinSAT: In the first approach, it defines three different cost-preserving clausal form transformations –called direct, improved and Tseitin-based transformations– from non-clausal MaxSAT/MinSAT to clausal MaxSAT/MinSAT. These transformations reduce non-clausal MaxSAT/MinSAT to clausal MaxSAT/MinSAT so that clausal MaxSAT/MinSAT solvers can be used to solve the MaxSAT problem of multisets of propositional formulas that are not necessarily in clausal form. In the second approach, it defines a genuine tableau calculus for non-clausal MaxSAT, proves its soundness and completeness, and describes how it can be extended to deal with hard and soft formulas. Moreover, it describes how the tableau calculus for non-clausal MaxSAT can be used to solve non-clausal MinSAT.

Chapter 5: Solving the Team Composition Problem in a Classroom

This chapter defines the Team Composition Problem in a Classroom (TCPC), proves that it is NP-hard, and defines two different MaxSAT models of the problem, called maximizing and minimizing encodings. It also reports on the results of an empirical investigation that shows that solving the TCPC as a MaxSAT problem is promising, and provides evidence that the minimizing encoding outperforms the maximizing encoding. Finally, it describes how the proposed approach can be extended to richer team formation problems, using the Synergistic Team Composition Model (STCM) problem as a case study.

Chapter 6: Conclusions and Future Work

This chapter summarizes the main contributions of the thesis, and outlines a few research directions that we plan to pursue in the future.

Chapter 2

Preliminaries

This chapter defines the MaxSAT and MinSAT problems and overviews the most relevant algorithms and techniques used to solve them. It first describes the main solving methods for exact MaxSAT: integer linear programming, MaxSAT resolution, branch-and-bound MaxSAT solvers, SAT-based MaxSAT solvers and implicit hitting set-based solvers. Then, it reviews some solving techniques for MinSAT and describes a branch-and-bound MinSAT solver. Moreover, it devotes a section to the MaxSAT Evaluation.

Some parts of this chapter closely follow the chapter on MaxSAT in the Handbook of Satisfiability [131]. The description of a MinSAT branch-and-bound solver closely follows [148].

2.1 The MaxSAT and MinSAT problems

Given a set of propositional variables $\{x_1, \ldots, x_n\}$, a literal is a variable x_i or its negation $\neg x_i$. A weighted clause is a pair (c, w), where c is a disjunction of literals and w, its weight, is a positive integer or infinity. If its weight is infinity, it is a hard clause (we omit infinity weights for simplicity); otherwise it is a soft clause.

A truth assignment assigns to each variable either 0 (false) or 1 (true). It satisfies literal x_i ($\neg x_i$) if x_i evaluates to 1 (0); it satisfies weighted clause (c, w) if it satisfies a literal of c; and it satisfies a multiset of clauses if it satisfies all its clauses. The weight w is the penalty of violating clause c. When all clauses have the same weight, their weights can be omitted.

The Weighted Partial MaxSAT problem, or WPMaxSAT, for a multiset of weighted clauses ϕ is to find an assignment that satisfies all the hard clauses and minimizes (maximizes) the sum of the weights of the unsatisfied (satisfied) soft clauses. The most common subproblems of WPMaxSAT are the following: Weighted MaxSAT (WMaxSAT), which is WPMaxSAT without hard clauses; Partial MaxSAT (PMaxSAT), which is WPMaxSAT when all the soft clauses have the same weight, and MaxSAT, which is PMaxSAT without hard clauses.

The Weighted Partial MinSAT problem, or WPMinSAT, for a multiset of weighted clauses ϕ is to find an assignment that satisfies all the hard clauses and maximizes (minimizes) the sum of the weights of the unsatisfied (satisfied) soft clauses. The most common subproblems of WPMinSAT are the following: Weighted MinSAT (WMinSAT), which is WPMinSAT without hard clauses; Partial MinSAT (PMinSAT), which is WPMinSAT when all the soft clauses have the same weight, and MinSAT, which is PMinSAT without hard clauses.

We defined SAT as the problem of deciding if there is an assignment that satisfies a given propositional formula in conjunctive normal form (CNF). SAT can also be defined as PMaxSAT/PMinSAT without soft clauses.

We represent MaxSAT and MinSAT instances as multisets of clauses. Repeated clauses cannot be collapsed into one clause as in SAT, because then the maximum/minimum number of unsatisfied clauses might not be preserved. For example, the multiset of unit clauses $\{x_1, \neg x_1, x_1, \neg x_1\}$ has a minimum of two unsatisfied clauses while $\{x_1, \neg x_1\}$ has just one unsatisfied clause. In fact, the multiset $\{x_1, \neg x_1, x_1, \neg x_1\}$, of unit clauses, is equivalent to the multiset of weighted clauses $\{(x_1, 2), (\neg x_1, 2)\}$. Clauses can be represented by the set of its literals as in SAT because repeated literals can be collapsed into one literal without affecting the preservation of the number of unsatisfied clauses.

A Weighted Partial MaxSAT instance is a multiset of weighted clauses

$$\phi = \{(h_1, \infty), \ldots, (h_k, \infty), (c_1, w_1), \ldots, (c_m, w_m)\},$$

where the first k clauses are hard and the last m clauses are soft. For simplicity, in what follows, we will write $\phi = \{h_1, \ldots, h_k, (c_1, w_1), \ldots, (c_m, w_m)\}$, omitting the infinity weights. A soft clause (c, w) is equivalent to having w copies of the clause $(c, 1)$, and $\{(c, w_1), (c, w_2)\}$ is equivalent to $(c, w_1 + w_2)$.

Let ϕ be a multiset of clauses and let $l_1, \ldots, l_r$ be literals that occur in ϕ. The instantiation of $l_1, \ldots, l_r$ in ϕ, denoted by $\phi_{l_1|\cdots|l_r}$, is the multiset of clauses resulting of eliminating from ϕ all the occurrences of $\neg l_1, \ldots, \neg l_r$ and removing all the clauses with occurrences of $l_1, \ldots, l_r$.

In MaxSAT/MinSAT, formulas are equivalent if they have the same amount of unsatisfied clauses for every assignment. Similarly, we say two formulas ϕ_1 and ϕ_2 are equivalent in Weighted MaxSAT if the sum of weights of the unsatisfied clauses is equal for every assignment for both formulas.

2.2 Complete MaxSAT algorithms

This section presents an overview of the most important complete MaxSAT algorithms and the solving techniques they implement. Section 2.2.1 defines the integer linear programming formulation of MaxSAT. Section 2.2.2 defines the MaxSAT resolution rule and explains how it can solve MaxSAT. Section 2.2.3 describes the branch-and-bound schema for MaxSAT and some improvements, including good quality lower bounds. Section 2.2.4 describes some representative SAT-based MaxSAT solvers. Section 2.2.5 describes two relevant implicit hitting set-based MaxSAT solvers. Section 2.2.6 gives a brief description of the MaxSAT Evaluation. Section 2.2.7 summarizes other remarkable contributions to MaxSAT solving.

2.2.1 Integer linear programming formulation of MaxSAT

MaxSAT can be solved with a Mixed Integer Programming (MIP) solver as Gurobi [91] or IBM CPLEX [72] if a proper Integer Linear Programming (ILP) formulation is provided as input.

Let $\phi = \{(C_1, w_1), \ldots, (C_m, w_m)\}$ be a Weighted MaxSAT instance over the propositional variables $x_1, \ldots, x_n$. With each propositional variable x_i, we associate a variable $y_i \in \{0, 1\}$ such that $y_i = 1$ if variable x_i is true and $y_i = 0$, otherwise. With each clause C_j, we associate a variable $z_j \in \{0, 1\}$ such that $z_j = 1$ if clause C_j is satisfied and $z_j = 0$, otherwise. Let I_j^+ be the set of indices of unnegated variables in clause C_j, and let I_j^- be the set of indices of negated variables in clause C_j. The ILP formulation of the Weighted MaxSAT instance ϕ is defined as follows:

$$\max F(y, z) = \sum_{j=1}^{m} w_j z_j$$

subject to

$$\sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) \geq z_j \quad j = 1, \ldots, m$$

$$y_i \in \{0, 1\} \quad i = 1, \ldots, n$$

$$z_j \in \{0, 1\} \quad j = 1, \ldots, m$$

Assume now that, with each clause C_j, we associate a variable $z_j \in \{0, 1\}$ such that $z_j = 1$ if clause C_j is unsatisfied and $z_j = 0$, otherwise. Then, the ILP formulation of the minimization version of Weighted MaxSAT for the instance ϕ is defined as follows:

$$\min F(y, z) = \sum_{j=1}^{m} w_j z_j$$

subject to

$$\sum_{i \in I_j^+} y_i + \sum_{i \in I_j^-} (1 - y_i) + z_j \geq 1 \quad j = 1, \ldots, m$$

$$y_i \in \{0, 1\} \quad i = 1, \ldots, n$$

$$z_j \in \{0, 1\} \quad j = 1, \ldots, m$$

Ansótegui and Gabàs [28] reported an extensive empirical investigation that indicates that solving MaxSAT instances by translating them into ILP and applying a MIP solver is competitive on crafted instances.

2.2.2 MaxSAT resolution

The MaxSAT resolution rule [63, 64, 93] can be seen as an extension of the SAT resolution rule that preserves the number of unsatisfied clauses between the premises and the conclusion. The SAT resolution rule is unsound for MaxSAT and MinSAT because it only preserves satisfiability, but the MaxSAT resolution rule is sound for both MaxSAT and MinSAT.

The MaxSAT resolution rule is defined as follows:

$$\begin{array}{l}
x \vee a_1 \vee \cdots \vee a_s \\
\overline{x} \vee b_1 \vee \cdots \vee b_t \\
\hline\hline
a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_t \\
x \vee a_1 \vee \cdots \vee a_s \vee \overline{b_1} \\
x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \overline{b_2} \\
\cdots \\
x \vee a_1 \vee \cdots \vee a_s \vee b_1 \vee \cdots \vee b_{t-1} \vee \overline{b_t} \\
\overline{x} \vee b_1 \vee \cdots \vee b_t \vee \overline{a_1} \\
\overline{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \overline{a_2} \\
\cdots \\
\overline{x} \vee b_1 \vee \cdots \vee b_t \vee a_1 \vee \cdots \vee a_{s-1} \vee \overline{a_s}
\end{array}$$

This inference rule concludes, apart from the conclusion where a variable has been cut, some additional clauses that contain one of the premises as subclause. We say that the rule *cuts* the variable x. The tautologies concluded by the rule are removed, and the repeated literals in a clause are collapsed into one. Notice that an instance of MaxSAT resolution not only depends on the two premises and the cut variable (like in resolution), but also on the order of the literals in the premises. Notice also that, like in resolution, this rule concludes a new clause not containing the variable x, except when this clause is a tautology.

Bonet et al. [63, 64] proved the completeness of MaxSAT resolution: By saturating successively w.r.t. all the variables, one derives as many empty clauses as the minimum number of unsatisfied clauses in the MaxSAT input instance. Saturating w.r.t. a variable amounts to apply the MaxSAT resolution rule to clauses containing that variable until every possible application of the inference rule only introduces clauses containing that variable (since tautologies are eliminated). Once a MaxSAT instance is saturated w.r.t. a variable, all the clauses containing that variable are not considered to saturate w.r.t. another variable. We refer to [64] for further technical details and the weighted version of the rule.

We consider the multiset of clauses $\phi = \{\overline{x}_1, x_1 \vee x_2, x_1 \vee x_3, \overline{x}_3\}$ to illustrate how a variable saturation algorithm works. We start by considering variable x_1 and resolve the first two clauses, obtaining $\{x_2, \overline{x}_1 \vee \overline{x}_2, x_1 \vee x_3, \overline{x}_3\}$. We then resolve the second and third clause and get a saturation of ϕ w.r.t. x_1: $\{x_2, \overline{x}_2 \vee x_3, \overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3, x_1 \vee x_2 \vee x_3, \overline{x}_3\}$. From now on, we only consider the clauses not containing x_1: $C_1 = \{x_2, \overline{x}_2 \vee x_3, \overline{x}_3\}$, and ignore the clauses containing x_1: $\{\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3, x_1 \vee x_2 \vee x_3\}$. We continue by resolving the first two clauses of C_1; we get $\{x_3, x_2 \vee \overline{x}_3, \overline{x}_3\}$, which is a saturation of

C_1 w.r.t. x_2. Hence, $C_2 = \{x_3, \overline{x}_3\}$ is the resulting multiset of clauses not containing x_2 and ignore $\{x_2 \vee \overline{x}_3\}$, which is the multiset of clauses containing x_2. Finally, we resolve $\{x_3, \overline{x}_3\}$ and get the empty clause. Since all the variables have been saturated, the minimum number of unsatisfied clauses in ϕ is 1.

The use of restrictions of MaxSAT resolution has not been limited to branch-and-bound solvers. Narodytska and Bacchus [172] used MaxSAT resolution in SAT-based MaxSAT solvers, avoiding the use of cardinality constraints and obtaining very competitive results on industrial instances.

There exists no polynomial-size resolution proof of the Pigeon Hole Principle (PHP). However, Ignatiev et al. [104] showed that there exist polynomial-size MaxSAT resolution proofs of PHP if PHP is encoded as a Partial MaxSAT instance using the dual rail encoding. Indeed, the combination of the dual rail encoding and MaxSAT resolution is a stronger proof system than either general resolution or conflict-driven clause learning [61]. We refer the reader to [62] for the latest proof complexity results related to MaxSAT resolution.

MaxSAT resolution has been extended to the multiple-valued clausal forms known as signed CNF formulas [50, 156, 157]. The defined signed MaxSAT resolution rules are complete and provide a logical framework for Weighted Constraint Satisfaction Problems (WCSP) [25]. Besides, some restrictions of the rules enforce the defined local consistency properties for WCSPs in a natural way [24, 26].

2.2.3 Branch-and-bound algorithms for MaxSAT

There are competitive exact MaxSAT solvers —as the ones developed by [3, 13, 15, 16, 33, 94, 122, 125, 138, 149, 150, 178, 180, 184, 194, 195, 201]— that implement variants of the following Branch-and-Bound (BnB) scheme: Given a MaxSAT instance ϕ, BnB explores the search tree that represents the space of all possible assignments for ϕ in a depth-first manner. At every node, BnB compares the upper bound (UB), which is the best solution found so far for a complete assignment, with the lower bound (LB), which is the sum of the number of clauses which are unsatisfied by the current partial assignment plus an underestimation of the number of clauses that will become unsatisfied if the current partial assignment is completed. If $LB \geq UB$, the algorithm prunes the subtree below the current node and backtracks chronologically to a higher level in the search tree. If $LB < UB$, the algorithm tries to find a better solution by extending the current partial assignment by instantiating one more variable. The optimal number of unsatisfied clauses in the input MaxSAT instance is the value that UB takes after exploring the entire search tree.

Figure 2.1 shows the pseudo-code of a basic BnB MaxSAT solver. It uses the following notation:

- *simplifyFormula*(ϕ) is a procedure that transforms ϕ into an equivalent and simpler instance by applying inference rules.
- $\#emptyClauses(\phi)$ is a function that returns the number of empty clauses in ϕ.

Algorithm 2.1: MaxSAT(ϕ, UB) : The MaxSAT branch-and-bound schema

```
Input: $\phi$ , UB        ▷ A CNF MaxSAT formula and an initial UB integer value

Function MaxSAT ($\phi : CNFformula, UB : upperbound$):
    $\phi \leftarrow simplifyFormula(\phi)$
    if $\phi = \emptyset \vee \phi$ only contains empty clauses then
        return #emptyClauses($\phi$)
    $LB \leftarrow \#emptyClauses(\phi) + underestimation(\phi)$
    if $LB \geq UB$ then
        return $UB$
    $x \leftarrow selectVariable(\phi)$
    $UB \leftarrow min(UB, MaxSAT(\phi_{\neg x}, UB))$
    return $min(UB, MaxSAT(\phi_x, UB))$
```

- LB is a lower bound of the minimum number of unsatisfied clauses in ϕ if the current partial assignment is extended to a complete assignment. We assume that its initial value is 0.

- *underestimation*(ϕ) is a function that returns an underestimation of the minimum number of non-empty clauses in ϕ that will become unsatisfied if the current partial assignment is extended to a complete assignment.

- UB is an upper bound of the number of unsatisfied clauses in an optimal solution. An elementary initial value for UB is the total number of clauses in the input formula, or the number of clauses which are unsatisfied by an arbitrary interpretation. Nevertheless, most of the solvers take as initial upper bound the number of unsatisfied clauses that can be detected by executing the input formula in a local search solver during a short period of time.

- *selectVariable*(ϕ) is a function that returns a variable of ϕ following an heuristic.

- ϕ_x ($\phi_{\bar{x}}$) is the formula obtained by setting the variable x to true (false); i.e., by applying the one-literal rule to ϕ using the literal x ($\bar{x}$).

Modern MaxSAT solvers implement the basic algorithm augmented with powerful inference techniques, good quality lower bounds, clever variable selection heuristics and efficient data structures. Partial MaxSAT solvers are also augmented with learning of hard clauses, and non-chronological backtracking.

Improving the lower bound with underestimations

The simplest method to compute a lower bound consists in just counting the number of clauses which are unsatisfied by the current partial assignment [65]. One step forward is to incorporate an underestimation of the number of clauses that will become unsatisfied if the current partial assignment is extended to a complete assignment. The most basic method was defined by Wallace and Freuder [191]:

$$\texttt{LB}(\phi) = \#emptyClauses(\phi) + \sum_{x \text{ occurs in } \phi} min(ic(x), ic(\bar{x})),$$

where ϕ is the CNF formula associated with the current partial assignment, and $ic(x)$ ($ic(\bar{x})$) —inconsistency count of x ($\bar{x}$)— is the number of unit clauses of ϕ that contain $\bar{x}$ (x). In other words, that underestimation is the number of disjoint inconsistent subformulas in ϕ formed by a unit clause with a literal l and a unit clause with the complementary of l.

Lower bounds dealing with longer clauses include the star rule and UP. In the star rule [14, 184], the underestimation of the lower bound is the number of disjoint inconsistent subformulas of the form $\{l_1, \ldots, l_k, \bar{l}_1 \vee \cdots \vee \bar{l}_k\}$. When $k = 1$, the star rule is equivalent to the inconsistency counts of Wallace and Freuder.

Given a multiset of clauses ϕ containing a unit clause $\{l\}$, the one-literal rule removes from ϕ the clauses containing l and removes the occurrences of $\neg l$ from the clauses in which $\neg l$ appears. The repeated application of the one-literal rule until there is no unit clause is known as unit propagation. Unit propagation can be computed in time linear in the size of the input instance.

In UP [136], the underestimation of the lower bound is the number of disjoint inconsistent subformulas that can be detected with unit propagation. UP works as follows: It applies unit propagation until a contradiction is derived. Then, UP identifies, by inspecting the implication graph, a subset of clauses from which a unit refutation can be constructed, and tries to identify new contradictions from the remaining clauses. The order in which unit clauses are propagated has a clear impact on the quality of the lower bound [137].

UP can be enhanced with failed literal detection as follows: Given a MaxSAT instance ϕ and a variable x occurring positively and negatively in ϕ, UP is applied to both $\phi \wedge \{x\}$ and $\phi \wedge \{\bar{x}\}$. If UP derives a contradiction from $\phi \wedge \{x\}$ and another contradiction from $\phi \wedge \{\bar{x}\}$, then the union of the two inconsistent subsets identified by UP, once we have removed the unit clauses x and $\bar{x}$, is an inconsistent subset of ϕ. UP enhanced with failed literal detection does not need the occurrence of unit clauses in the input formula for deriving a contradiction. While UP only identifies unit refutations, UP enhanced with failed literal detection identifies non-unit refutations too. Since applying failed literal detection to every variable is time-consuming, it is applied to a reduced number of variables in practice [137].

MaxSAT solvers like ahmaxsat [3], MaxSatz [138], and MiniMaxSat [94] apply either UP or UP enhanced with failed literal detection. Nowadays, UP-based lower bounds are the prevailing approach to computing underestimations in BnB MaxSAT solvers. This technique has also been applied to solve the maximum clique problem [112, 128, 129, 130, 146].

Darras et al. [75] developed a version of UP in which the computation of the lower bound is made more incremental by saving some of the small size disjoint inconsistent subformulas detected by UP. They avoid to redetect the saved inconsistencies if they remain in subsequent nodes of the proof tree, and are able to solve some types of instances faster. Lin et al. [150] defined an improved version of UP that, besides being incremental, guarantees that the lower bound computed at a node of the search tree is not smaller than the lower bound computed at the parent of that node. Abramé and Habet [5] proposed an improved implementation of UP in ahmaxsat that undoes propagations in non-chronological order and can produce smaller inconsistent subsets. Shen and Zhang [184] defined a lower bound, called LB4, which is similar to UP but restricted to Max-2SAT instances and using a static variable ordering.

A variant of UP enhanced with failed literal detection was implemented in the solver akmaxsat [122]. It can be the case that UP derives a contradiction from $\phi \wedge \{l\}$ but not from $\phi \wedge \{\neg l\}$. In fact, this shows that $\neg l$ follows from ϕ. If ϕ' is the result of applying UP to $\phi \wedge \{\neg l\}$, then the algorithms tries to find another failed literal l' in ϕ'. If UP derives a contradiction from both $\phi \wedge \{l'\}$ and $\phi \wedge \{\neg l'\}$, the algorithm stops and identifies an inconsistent subset. If UP derives a contradiction from $\phi \wedge \{l'\}$ but not from $\phi \wedge \{\neg l'\}$, the same process is repeated on the formula resulting of applying UP to $\phi \wedge \{\neg l'\}$ until an inconsistent subset is detected or no more failed literals can be found.

Improving the lower bound with inference

Another approach to improve the quality of the lower bound consists in applying inference rules that transform a MaxSAT instance ϕ into an equivalent but simpler MaxSAT instance ϕ'. In the best case, inference rules produce new empty clauses in ϕ' that allow incrementing the lower bound. In contrast with the empty clauses derived when computing underestimations, the empty clauses derived with inference rules do not have to be recomputed at every node of the current subtree so that the lower bound computation is more incremental.

Unfortunately, unit propagation, which is the most powerful inference technique applied in DPLL-style SAT solvers, is unsound for MaxSAT as the next example shows: The set of clauses $\{x_1, \neg x_1 \vee x_2, \neg x_1 \vee \neg x_2, \neg x_1 \vee x_3, \neg x_1 \vee \neg x_3\}$ has a minimum of one unsatisfied clause (setting x_1 to false), but two empty clauses are derived by applying unit propagation.

The amount of inference enforced by existing BnB MaxSAT solvers at each node of the proof tree is poor compared with the inference enforced by DPLL-style SAT solvers. The simplest inference enforced, when branching on a literal l, consists in applying the one-literal rule: The clauses containing l are removed from the instance and the occurrences of $\neg l$ are removed from the clauses in which $\neg l$ appears, but the existing unit clauses and the new unit clauses derived as a consequence of removing the occurrences of $\neg l$ are not propagated as in unit propagation. That inference is typically enhanced with the MaxSAT inference rules described in the rest of this section.

First, we present simple inference rules that have proved to be useful in a number of solvers [13, 15, 65, 184, 195], and then some more sophisticated inferences rules which are implemented in solvers like ahmaxsat [3], akmaxsat [122], MaxSatz [138], and MiniMaxSat [94]. Some simple inference rules are:

- The pure literal rule [65]: If a literal only appears with either positive or negative polarity in a MaxSAT instance, all the clauses containing that literal are removed.
- The dominating unit clause rule [173]: If the number of clauses (of any length) in which a literal l appears is not greater than the number of unit clauses in which $\neg l$ appears, all the clauses containing l and all the occurrences of $\neg l$ are removed.
- The complementary unit clause rule [173]: If a MaxSAT instance contains a unit clause with the literal l and a unit clause with the literal $\neg l$, these two clauses are replaced with one empty clause.
- The almost common clause rule [49]: If a MaxSAT instance contains a clause $x \vee D$ and a clause $\neg x \vee D$, where D is a disjunction of literals, then both clauses are replaced with D. In practice, this rule is applied when D contains at most one literal.

We present now refinements of the MaxSAT resolution rule that can be applied in polynomial time and mitigate the impossibility of applying unit propagation in MaxSAT solving. We start by presenting the *star rule*: If ϕ_1={l_1, $\bar{l}_1 \vee \bar{l}_2$, l_2}$\cup\phi'$, then ϕ_2={$\Box$, $l_1 \vee l_2$} $\cup\ \phi'$ is equivalent to ϕ_1. This rule can also be presented as follows:

$$\left\{ \begin{array}{c} l_1 \\ \neg l_1 \vee \neg l_2 \\ l_2 \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} \Box \\ l_1 \vee l_2 \end{array} \right\} \tag{2.1}$$

Notice that the rule detects a contradiction from $l_1, \bar{l}_1 \vee \bar{l}_2, l_2$ and, therefore, replaces these clauses with an empty clause. In addition, the rule adds the clause $l_1 \vee l_2$ to ensure the equivalence between ϕ_1 and ϕ_2. For any assignment containing either $l_1 = 0, l_2 = 1$, or $l_1 = 1, l_2 = 0$, or $l_1 = 1, l_2 = 1$, the number of unsatisfied clauses in $\{l_1, \bar{l}_1 \vee \bar{l}_2, l_2\}$ is 1, but for any assignment containing $l_1 = 0, l_2 = 0$, the number of unsatisfied clauses is 2. Notice that even when any assignment containing $l_1 = 0, l_2 = 0$ is not the best assignment for the subset $\{l_1, \bar{l}_1 \vee \bar{l}_2, l_2\}$, it can be the best for the whole formula. By adding $l_1 \vee l_2$, the rule ensures that the number of unsatisfied clauses in ϕ_1 and ϕ_2 is also the same when $l_1 = 0, l_2 = 0$.

This rule can be generalized in such a way that it captures unit resolution refutations in which clauses and resolvents are used exactly once:

$$\left\{ \begin{array}{c} l_1 \\ \bar{l}_1 \vee l_2 \\ \bar{l}_2 \vee l_3 \\ \cdots \\ \bar{l}_k \vee l_{k+1} \\ \bar{l}_{k+1} \end{array} \right\} \Longrightarrow \left\{ \begin{array}{c} \Box \\ l_1 \vee \bar{l}_2 \\ l_2 \vee \bar{l}_3 \\ \cdots \\ l_k \vee \bar{l}_{k+1} \\ \\ \end{array} \right\} \tag{2.2}$$

The last two rules consume two unit clauses for deriving one contradiction. Next, we define two inference rules that capture unit resolution refutations in which (i) exactly one unit clause is consumed, and (ii) the unit clause is used twice in the derivation of the empty clause. The second rule is a combination of the first rule with a linear derivation.

$$\left\{\begin{array}{c} l_1 \\ \bar{l}_1 \vee l_2 \\ \bar{l}_1 \vee l_3 \\ \bar{l}_2 \vee \bar{l}_3 \end{array}\right\} \Longrightarrow \left\{\begin{array}{c} \Box \\ l_1 \vee \bar{l}_2 \vee \bar{l}_3 \\ \bar{l}_1 \vee l_2 \vee l_3 \end{array}\right\} \tag{2.3}$$

$$\left\{\begin{array}{c} l_1 \\ \bar{l}_1 \vee l_2 \\ \bar{l}_2 \vee l_3 \\ \cdots \\ \bar{l}_k \vee l_{k+1} \\ \bar{l}_{k+1} \vee l_{k+2} \\ \bar{l}_{k+1} \vee l_{k+3} \\ \bar{l}_{k+2} \vee \bar{l}_{k+3} \end{array}\right\} \Longrightarrow \left\{\begin{array}{c} \Box \\ l_1 \vee \bar{l}_2 \\ l_2 \vee \bar{l}_3 \\ \cdots \\ l_k \vee \bar{l}_{k+1} \\ l_{k+1} \vee \bar{l}_{k+2} \vee \bar{l}_{k+3} \\ \bar{l}_{k+1} \vee l_{k+2} \vee l_{k+3} \end{array}\right\} \tag{2.4}$$

MaxSatz implements the almost common clause rule, Rule 2.1, Rule 2.2, Rule 2.3 and Rule 2.4. Some of these rules are also applied in the solvers ahmaxsat and akmaxsat. We refer the reader to [133, 134, 135] for an empirical analysis of different inference rules.

The lower bound computation methods based on unit propagation represent the different derivations of unit clauses in a graph, called implication graph [138]. Looking at that graph, solvers identify the clauses which are involved in the derivation of a contradiction. In contemporary MaxSAT solvers, this graph is also used to decide whether the clauses involved in a contradiction match with the premises of the above mentioned inference rule.

Abramé and Habet [7] showed that, in some cases, it is better not to apply MaxSAT resolution to certain inconsistent subsets of clauses because the transformations derived do not allow to detect further inconsistent subsets. They also showed that, in other cases, further inconsistent subsets could be detected if MaxSAT resolution is applied locally [4].

Variable selection heuristics

Most of the exact MaxSAT solvers incorporate variable selection heuristics that take into account the number of literal occurrences in such a way that each occurrence has an associated weight that depends on the length of the clause that contains the literal.

MaxSAT heuristics give priority to literals occurring in binary clauses instead of literals occurring in unit clauses as SAT heuristics do.

Let us see as an example the variable selection heuristic of MaxSatz [138]: Let $neg1(x)$ ($pos1(x)$) be the number of unit clauses in which x is negative (positive), $neg2(x)$($pos2(x)$) be the number of binary clauses in which x is negative (positive), and let $neg3(x)$ ($pos3(x)$) be the number of clauses containing three or more literals in which x is negative (positive). MaxSatz selects the variable x such that ($neg1(x) + 4 * neg2(x) + neg3(x)$)*($pos1(x) + 4 * pos2(x) + pos3(x)$) is the largest. Once a variable x is selected, MaxSatz applies the following value selection heuristic: If $neg1(x) + 4 * neg2(x) + neg3(x) < pos1(x) + 4 * pos2(x) + pos3(x)$, set x to true; otherwise, set x to false. The solver ahmaxsat [3] implements a variant of this variable selection heuristic.

Earlier MaxSAT solvers, such as Lazy [15], MaxSolver [195], Max-DPLL [125], AMP [13] and others, incorporate variants of the two-sided Jeroslow rule that give priority to variables often occurring in binary clauses.

Data structures

Data structures for SAT have been naturally adapted to MaxSAT. We can divide the solvers into two classes: solvers like ahmaxsat, akmaxsat and MaxSatz representing formulas with adjacency lists, and solvers like Lazy and MiniMaxSat which use data structures with watched literals. Lazy data structures are particularly useful when there is a big number of clauses; for example, in Partial MaxSAT solvers with clause learning.

2.2.4 SAT-based MaxSAT algorithms

SAT-based MaxSAT algorithms proceed by reformulating the MaxSAT optimization problem into a sequence of SAT decision problems. Each SAT instance of the sequence encodes whether there exists an assignment to the MaxSAT instance with a cost less than or equal to a certain k. SAT instances with a k less than the optimal cost are unsatisfiable, while the others are satisfiable. The SAT solver is executed in incremental mode in order to keep the clauses learnt at each iteration over the sequence of SAT instances. There are two main types of SAT-based MaxSAT solvers: model-guided and core-guided. Model-guided MaxSAT solvers iteratively refine (decrease) the upper bound and guide the search with satisfying assignments obtained from satisfiable SAT instances. Core-guided MaxSAT solvers iteratively refine (increase) the lower bound and guide the search with the unsatisfiable cores obtained from unsatisfiable SAT instances. Both have strengths and weaknesses, also existing hybrid approaches. Representative solvers of this group are msu1.2 [161, 162], WBO [153, 154], Open-WBO [166], WPM1 [21], PM2 [23], WPM2 [22], WPM3 [29], Eva [172], SAT4J-Maxsat [54, 55], QMaxSat [121, 198] and Pacose [84].

The Fu and Malik algorithm [86] is among the first and most relevant SAT-based Partial MaxSAT algorithms. It was implemented in the solver msu1.2 [161, 162] and its correctness was proved in [21]. Algorithm 2.2 shows its pseudo-code. It calls a

SAT solver (line 5) in an iterative manner, modifying the Partial MaxSAT instance $\phi = \{h_1, \ldots, h_n,\ (s_1, 1), \ldots, (s_m, 1)\}$ in each iteration. If ϕ is satisfiable, the SAT solver returns true (SAT). If ϕ is unsatisfiable, the SAT solver returns false (UNSAT) and an unsatisfiable core ϕ_c, which is a set of soft clauses that is unsatisfiable when combined with the hard clauses. Then, the algorithm modifies the working formula ϕ by adding new variables, known as blocking variables (BV), to each soft clause in the core ϕ_c. The formula ϕ is also modified by the addition of hard clauses, corresponding to the CNF encoding of a cardinality constraint stating that exactly one of the new blocking variables must be true. At this point, at least one of the soft clauses in the core will be unsatisfied. The counter of unsatisfied clauses, which provides a lower bound, is increased by one. The algorithm stops when the SAT solver returns true (SAT).

Algorithm 2.2: FuMalik(ϕ): The Fu and Malik algorithm

```
Input: $\phi = \{h_1, \ldots, h_n,\ (s_1, 1), \ldots, (s_m, 1)\}$

1  if $\neg SAT(\{h_i \in \phi\})$ then
2      return $(\infty, \emptyset)$                          ▷ Hard clauses can not be satisfied
3  $cost = 0$
4  while true do
5      $(st, \phi_c) = SAT(\{h_i \in \phi\} \cup \{s_i \mid (s_i, 1) \in \phi\})$      ▷ Removed weights
6      if st = SAT then
7          return $(cost, \phi)$
8      $BV = \emptyset$                                       ▷ Set of blocking variables
9      foreach $s_i \in \phi_c$ do
10         $b =$ new_blocking_variable()
11         $\phi = \phi \setminus \{s_i\} \cup \{(s_i \vee b, 1)\}$           ▷ Blocking variable addition
12         $BV = BV \cup \{b\}$
13 $\phi = \phi \cup CNF(\sum_{b \in BV} b = 1)$          ▷ Cardinality constraint to hard clauses
14 $cost = cost + 1$
```

The Fu and Malik algorithm has been extended to Weighted Partial MaxSAT in solvers as WBO [153, 154] and WPM1 [21]. WBO uses a pseudo-Boolean solver instead of a SAT solver and so the cardinality constraint $\sum_{b \in BV} b = 1$ does not need to be encoded to CNF. WBO and WPM1 update the cost by adding the minimum weight of the soft clauses in the core ϕ_c. Moreover, every soft clause in ϕ_c is replaced by two copies: an extended one with an additional auxiliary variable and whose weight is set to the minimum weight of the soft clauses in ϕ_c, and an unextended one whose weight is set to the original weight minus the minimum weight in ϕ_c.

The Fu and Malik algorithm adds a blocking variable for each soft clause in a given core. Depending on the instance, the number of blocking variables added might be large, thus hampering the SAT solver performance in successive calls. The Partial MaxSAT solver PM2 [23] deals with this problem by adding only one blocking variable to each soft clause in the input instance.

Algorithm 2.3 shows the pseudo-code of PM2. The set $BV = \{b_1, \ldots, b_m\}$ of blocking variables are defined and added to the soft clauses of the original formula. The working instance becomes $\phi_w = \{h_1, \ldots, h_n,\ s_1 \vee b_1, \ldots, s_m \vee b_m\}$, where clause weights are discarded. The list AL contains at-least constraints over blocking variables. The list L contains the discovered cores, in the form of sets of indexes corresponding to the soft clauses involved. Before starting to iterate the algorithm, the counter $cost$ of unsatisfied clauses is set to zero, this is the lower bound. The lists L and AL are initialized to the empty set. At this point, we need to define the concept of cover: given a set of cores L, we say that the set of indexes B is a $cover$ of L if it is a minimal non-empty set such that, for every $A \in L$, if $A \cap B \neq \emptyset$, then $A \subseteq B$.

A partition of L in disjoint covers ($SC(L)$) is incrementally updated in each iteration although, for simplicity reasons, the pseudo-code in Algorithm 2.3 shows that $SC(L)$ is re-calculated from scratch for each iteration. The list of at-most constraints AM always contains a cardinality constraint for every cover $B \in SC(L)$, stating that the sum of the blocking variables of the cover must be at most equal to the number of cores contained in the cover. The SAT solver calls in PM2 are carried out over the formula $\phi_w \cup CNF(AL \cup AM)$. If the SAT solver returns true (SAT), the algorithm returns $cost$ as the optimal and terminates. Otherwise, a core ϕ_c is returned, from which hard clauses can be removed.

Next, the set of indexes of the soft clauses in the new core (A) is appended to L. Now, the number of cores in L, such that their soft clauses indexes are included in A, is counted (including A) and stored in k. An at-least cardinality constraint, saying that the number of indexed blocking variables in A that need to be one is at least k, is appended to AL. Finally, the cost is incremented by one because of the new core found. The algorithm continues iterating until the SAT solver returns true (SAT).

The WPM2 solver [22], which is the weighted version of PM2, deals with weighted instances by using pseudo-Boolean constraints instead of cardinality constraints, thus avoiding to maintain two copies of each soft clause that appears in a newly discovered core. The WPM2 solver is also able to increase the lower bound by more than one at each iteration by optimizing subproblems. This subproblem solving is known as cover optimization [20]. In the 2015 MaxSAT Evaluation, the WPM3 solver [29] was presented. WPM3 performs cover optimization for subproblems as WPM2. Moreover, WPM3 is also inspired by Eva [172] and OLL [19], which take advantage of the so-called MaxSAT reducibility by using different approaches. Given the Weighted Partial MaxSAT formulas ϕ and ϕ', we say ϕ is MaxSAT reducible to ϕ' if, for any of the assignments $I : var(\phi) \rightarrow \{0, 1\}$, it holds that $I(\phi) = min\{I'(\phi') \mid I'(x) = I(x)\ \forall x \in var(\phi)\}$.

Model-based MaxSAT solvers as SAT4J-Maxsat [54, 55], QMaxSat [121] and Pacose [84] start with a satisfiable formula where $cost \leq \sum_{i=1}^{m} w_i$, and decrease this value until the SAT solver returns unsatisfiable. The latest model found is the solution and its cost is the optimal.

QMaxSat [121] is a model-based Partial MaxSAT solver that uses an underlying

Algorithm 2.3: PM2(ϕ): The pseudo-code of the PM2 algorithm

```
Input: $\phi = \{h_1, \ldots, h_n, \ (s_1, 1), \ldots, (s_m, 1)\}$

1  if $SAT(\{h_i \in \phi\}) = (UNSAT, \_)$ then
2      return $(\infty, \emptyset)$                                  ▷ Hard clauses are unsatisfiable
3  $BV = \{b_1, \ldots, b_m\}$                                        ▷ Set of blocking variables
4  $\phi_w = \{h_1, \ldots, h_n\} \cup \{s_1 \vee b_1, \ldots, s_m \vee b_m\}$
5  $cost = 0$
6  $L = \emptyset$                                                     ▷ Set of cores
7  $AL = \emptyset$                                                    ▷ Set of at-least constraints
8  while true do
9      $AM = \emptyset$                                                ▷ Set of at-most constraints
10     foreach $B \in SC(L)$ do
11         $k = |\{A \in L \mid A \subseteq B\}|$                         ▷ Num. of cores inside cover B
12         $AM = AM \cup \{\sum_{i \in B} b_i \leq k\}$
13     $(st, \phi_c) = SAT(\phi_w \cup CNF(AL \cup AM))$                  ▷ SAT solver call
14     if st = SAT then
15         return $(cost, \phi_w \cup CNF(AL \cup AM))$
16     $A = \{i \in \{1, \ldots, m\} \mid s_i \vee b_i \in \phi_c \wedge b_i \in BV\}$   ▷ $\phi_c$ soft c. indexes
17     $L = L \cup \{A\}$
18     $k = |\{A' \in L \mid A' \subseteq A\}|$        ▷ Num. of cores contained in A including A
19     $AL = AL \cup \{\sum_{i \in A} b_i \geq k\}$
20     $cost = cost + 1$
```

SAT solver and cardinality constraints encoded as defined in [48]. There is a Weighted Partial MaxSAT version of QMaxSat [198], which uses pseudo-Boolean instead of cardinality constraints. Pacose is based on the Weighted Partial QMaxSAT solver. SAT4J-Maxsat is a model-based Weighted Partial MaxSAT solver that has the ability to use a pseudo-Boolean solver instead of a SAT solver.

When the SAT or pseudo-Boolean solvers in QMaxSat, Pacose and SAT4J-Maxsat find a model, the satisfying assignment is checked, and the upper bound is set to the sum of weights of the soft clauses whose auxiliary blocking variable are true. At this point, a pseudo-Boolean constraint is added to the working formula, stating that any future model that the solver gets should fulfill that the *cost* value is strictly lower than the new upper bound.

Finally, the msu4.0 [163], pwbo1.1 [165] and bincd [95] solvers follow the hybrid approach. They search non-linearly for the optimal *cost* value, usually by a dichotomous search, alternating phases in which the SAT solver reports satisfiable (giving an upper bound) with others in which it reports unsatisfiable (giving a lower bound). They also use a unique auxiliary variable. The efficiency of these solvers critically depends on the SAT solver used and the encodings of the cardinality and pseudo-Boolean constraints.

2.2.5 The implicit hitting set approach for MaxSAT

The Implicit Hitting Set (IHS) approach [77] provides one of the most effective ways of solving MaxSAT. In fact, IHS-based solvers have been among the top-performing solvers on Weighted MaxSAT instances since the 2016 MaxSAT Evaluation.

The main advantage of the IHS approach is that it clearly separates optimization and propositional reasoning (core extraction). Optimization is carried out by a separate minimum cost hitting set solver, usually a MIP solver, which is good on arithmetic reasoning. Propositional reasoning is carried out by a SAT solver, which checks the satisfiability of subsets of the original problem that are returned by the MIP solver. This separation allows specialization, achieving remarkable performance improvements. The most representative solvers implementing the IHS approach are LMHS [183] and MaxHS [46, 53, 76, 77, 78, 79, 100].

Before describing MaxHS, we recall that a core k for a MaxSAT instance ϕ is a subset of soft clauses such that $k \cup hard(\phi)$ is unsatisfiable, where $hard(\phi)$ is the set of hard clauses of ϕ. Given a set of cores K, a hitting set hs of K is a set of soft clauses such that $hs \cap k \neq \emptyset$ for all $k \in K$. We say that hs is a minimum cost hitting set of K if it is a hitting set whose cost is less than or equal to the cost of any other hitting set of K.

Algorithm 2.4 shows the basic schema of MaxHS. The algorithm starts with an empty set of cores K. Each stage of the algorithm computes a minimum cost hitting set $\{s_1, \ldots, s_n\}$ and calls a SAT solver to determine if the multiset of clauses ϕ, taking off the soft clauses of the hitting set hs, is satisfiable. If ϕ is satisfiable, the SAT solver returns $(true, k)$ with k set to a satisfying assignment for $\phi \setminus hs$, otherwise the SAT solver returns $(false, k)$ with k set to a core of $\phi \setminus hs$. A satisfying assignment causes the algorithm to terminate, while an unsatisfying assignment adds a new core to K.

Algorithm 2.4: MaxHS-basic(ϕ): The most basic version of the MaxHS algorithm

```
Input: $\phi = \{h_1, \ldots, h_n,\ (s_1, w_1), \ldots, (s_m, w_m)\}$

1 $K = \emptyset$
2 while true do
3     $hs = FindMinCostHittingSet(K)$
4     $(sat, k) = SatSolver(\phi \setminus hs)$
5     if sat then
6         break
7     $K = K \cup \{k\}$
8 return ($\phi \setminus hs$, cost($hs$))
```

The behaviour of Algorithm 2.4 is influenced by three factors: the time required by the SAT solver to solve $\phi \setminus hs$, the time required by the MIP solver to solve the minimum cost hitting set problem, and the number of iterations in the loop. One of the most important improvements to Algorithm 2.4 is the minimization of cores before they are added to K. Adding a non-minimized core avoids that the last returned hs is obtained again. On the other hand, adding minimal cores to K, obtained by minimizing to an irreducible core, avoids the last hs returned by the MIP solver, but also a potentially huge amount of other cores that have not been returned yet. Minimization greatly reduces the number of loop iterations, calls to the SAT solver, and also to the MIP solver.

The empirical investigation in [79] provides evidence that the SAT solving time is typically negligible, and MaxHS performance is mainly affected by the solving time of the MIP solver and the number of iterations carried out by the algorithm. An obvious approach to reducing the time used by the MIP solver is the reduction of the number of calls to it. A simple method to avoid calling the MIP solver is the use of non-optimal approaches. These methods can provide approximate hitting sets hs, and if the later SAT solver call over $\phi \backslash hs$ returns false (unsatisfiable), they can be used instead of the optimal hitting sets. Two approximate methods were introduced in [79]. The first method is incremental and simply adds a clause to the newest minimal core of the current hitting set. Although the added clause can be any clause in the new core, it is preferred the clause that appears most frequently in the set of cores found so far. Thus, new cores will intersect with less already known cores. The second method is the use of a standard greedy algorithm for the hitting set problem [113], which ignores the current hitting set. This method usually returns a better hitting set than the first method.

Algorithm 2.5 describes MaxHS as in [79]. A two-level approach is used: first, the cheap incremental approximate method gives hitting sets until the SAT solver returns true (satisfiable); then, $nonOptLevel$ is assigned one. The greedy method becomes responsible of the task of returning a hitting set for once. If the SAT solver returns false (unsatisfiable), the returned core k is minimized and appended to K, and $nonOptLevel$ is assigned zero again (line 25), returning to the use of the cheap incremental method. Otherwise, if the greedy procedure also fails giving a hitting set such that $\phi \setminus hs$ is unsatisfiable, the inner while loop terminates and the algorithm iterates the outer loop. The MIP solver is again used to get an optimal hitting set (line 3) and $nonOptLevel$ is assigned zero (line 9).

Another improvement to the basic Algorithm 2.4 was introduced in [78] . The SAT solver first computes a set of disjoint cores, as in Algorithm 2.5 (line 1). This can only be easily done before starting the main loop.

Other approaches to reduce the time invested by the MIP solver, described in [78], include a method based on using non-core linear constraints to seed the MIP solver in the preprocessing phase, thus limiting the number of possible values of hs that can be obtained. Given the set $S = \{s_1, \ldots, s_m\}$ of soft clauses in ϕ, ϕ^b_{eq} is a relaxation of ϕ where the equivalence variables $B = \{b_1, \ldots, b_m\}$ are introduced and $\neg b_i \leftrightarrow s_i$ is enforced for all $i \in \{1, \ldots, m\}$. In this setting, a non-core constraint for a MaxSAT instance ϕ is a linear inequality constraint c, over equivalence auxiliary variables, such that $\phi^b_{eq} \models c$. Now, the MIP solver no longer solves a pure minimal hitting set problem because the seeded non-core constraints can contain negative b-variables. Algorithm 2.6 shows the non-core version of MaxHS.

There are some proposals to get non-core constraints in [78], and [79] recommends

Algorithm 2.5: MaxHS-nonOPT(ϕ): MaxHS using approximate hitting sets

```
Input: $\phi = \{h_1, \ldots, h_n,\ (s_1, w_1), \ldots, (s_m, w_m)\}$

$K = DisjointCores(\phi)$
while true do
    $hs = FindMinCostHittingSet(K)$                ▷ Get optimal solution
    $(sat, k) = SatSolver(\phi \setminus hs)$
    if sat then
        break
    $k = Minimize(k)$
    $K = K \cup \{k\}$
    $nonOptLevel = 0$
    while true do
        switch $nonOptLevel$ do
            case 0 do
                $hs = FindIncrementalHittingSet(K, k, hs)$
            case 1 do
                $hs = FindGreedyHittingSet(K)$
        if sat then
            switch $nonOptLevel$ do
                case 0 do
                    $nonOptLevel = 1$
                case 1 do
                    break
        else
            $k = Minimize(k)$
            $K = K \cup \{k\}$
            $nonOptLevel = 0$
return $(\phi \setminus hs$, cost$(hs))$
```

the use of *Eq-Seeding* among them because it was found to be the most effective overall. Eq-Seeding exploits equivalence relations between the literals that appear in soft unit clauses in the original MaxSAT instance ϕ and the corresponding b-variables. For each clause c in ϕ^b_{eq}, if each literal in c has an equivalent b-variable (with proper polarity), a new constraint can be derived from c by replacing each original literal in c by its equivalent b-variable. This constraint is a clause over the b-variables that can be added to the MIP solver. Given that the negations of the b-variables in ϕ^b_{eq} are equivalent to the corresponding soft clauses, the Eq-Seeding condition is obviously met for the original unit soft clauses of ϕ, but the constraint that could be added is already in ϕ^b_{eq}. However,

Algorithm 2.6: MaxHS-nO-seed(ϕ): MaxHS using approximate hitting sets and non-core constraints seeding

```
    Input: $\phi = \{h_1, \ldots, h_n,\ (s_1, w_1), \ldots, (s_m, w_m)\}$

1   $K = DisjointCores(\phi)$
2   $N = NonCoreConstraints(\phi^b_{eq})$
3   $obj = wt(c_i) * b_i + \cdots + wt(c_k) * b_k$
4   while true do
5       $A = Optimize(K \cup N, obj)$                 ▷ Get optimal solution
6       $(sat, k) = AssumptionSatSolver(\phi^b, A)$
7                                   ▷ Next lines are identical to Algorithm 2.5
```

taking into account the equivalences between the negation of b-variables and the original unit soft clauses, the literals in unit soft clauses of ϕ can be substituted in the non-unit clauses by its equivalent b-literals in ϕ^b_{eq}.

Let $\phi = \{(x), (\neg x), (x \vee y), (\neg y), (\neg x \vee z), (\neg z \vee y)\}$, a multiset of soft clauses having weight one. Then $\phi^b_{eq} = \{(b_1 \vee x), (b_2 \vee \neg x), (b_3 \vee x \vee y), (b_4 \vee \neg y), (b_5 \vee \neg x \vee z), (b_6 \vee \neg z \vee y)\} \cup CNF(b_1 \leftrightarrow x) \cup CNF(b_2 \leftrightarrow \neg x) \cup CNF(b_3 \leftrightarrow (x \vee y)) \cup CNF(b_4 \leftrightarrow \neg y) \cup CNF(b_5 \leftrightarrow (\neg x \vee z)) \cup CNF(b_6 \leftrightarrow (\neg z \vee y))$. It holds that $b_1 \equiv \neg x$, due to the soft unit clause (x) and its relaxation by b_1. Similarly, $b_4 \equiv y$. Therefore, from the relaxed clause $(b_3 \vee x \vee y) \in \phi^b_{eq}$, we obtain the b-variable hard constraint $(b_3 \vee \neg b_1 \vee b_4)$ by substituting the original literals by the equivalent b-variable literals.

Other methods for the generation of non-core constraints, described in [78], include *Imp-Seeding* and *Imp+Rev-Seeding*. The first method codifies in a unique linear inequality a set of implications from a b-literal to each of the other b-literals. Thus, a linear inequality is added for each of the b-literals if some implication is found for it. Given a b-literal b_i that implies k other b-literals, the inequality $-k \times b_i + b_i^1 + \cdots + b_i^k \geq 0$ codifies all the implications in a compact manner. The second method includes the first, but it also takes advantage of the set of b-literals that imply each of the other b-literals. The reverse implication sets for the second method can be obtained without additional effort from the implication sets of the first one. Thus, the Eq-Seeding method is used alone or combined with Imp-Seeding or Imp+Rev-Seeding.

In recent years, MaxHS has significantly improved its performance. This improvement is due to the reduced cost fixing technique of integer programming [46], the speeding-up of the embedded assumption-based SAT solver [100] and the introduction of the abstract cores, which provides a compact representation for a potentially exponential number of regular cores [53].

2.2.6 The MaxSAT Evaluation

There is an annual international evaluation of MaxSAT solvers since 2006 [35, 36, 37]. This event has triggered a tremendous acceleration in the development of new solvers and techniques for MaxSAT. The evaluation is divided into two main tracks, corresponding to complete and incomplete solvers. In the following, we focus on the complete track.

Until 2016, the MaxSAT Evaluation consisted of four tracks corresponding to the following problems: Unweighted, Weighted, Partial and Weighted Partial MaxSAT. Each of these tracks was divided into three groups corresponding to the following kinds of instances: randomly generated, crafted and industrial instances.

The solvers that participated until 2014 can be classified into two main groups: BnB and SAT-based MaxSAT solvers. Among the BnB solvers, we find: IncWMaxSatz [150], WMaxSatz [135, 138], akmaxsat [122], akmaxsat_ls [122] and ahmaxsat [3]. Among the SAT-based solvers, we find: PM2 [21], WPM1 [21], WPM2 [22], WPM3 [29], Eva [172] , wbo 1.4a [153], wbo 1.4b [154], SAT4J-Maxsat [54, 55] and QMaxSAT [121]. A third group was introduced in 2015: IHS solvers, being MaxHS [46, 77, 78, 79, 100] and LHMS [183] the first IHS solvers in the evaluation.

Since 2017, the MaxSAT Evaluation has two main tracks: the weighted track and the unweighted track. The (un)weighted track combines the industrial and crafted (un)weighted and (un)weighted partial MaxSAT categories from previous MaxSAT evaluations. Purely randomly generated instances are no longer evaluated.

Nine solvers competed in the 2020 MaxSAT Evaluation: EvalMaxSAT [45], MaxHS, Pacose [176], QMaxSAT [198], UWrMaxSat [177], Open-WBO [166], smax [155], Maxino [17] and RC2 [106]. All these solvers are essentially core-based, except for QMAXSAT and Pacose, which are model-based solvers, and MaxHS, which is an IHS solver. Table 2.1 classifies all solvers and indicates the three top-ranked solvers in the weighted and unweighted tracks. The core-based approach is referred to as Unsat-based (unsat-sat), while the model-based approach is referred to as Sat-Unsat. UWrMaxSat is classified as Unsat-based, but also as Sat-Unsat, because the solver can switch its behavior when convenient.

We reproduce now a summary of the information in [84] for the best ranked solvers in the unweighted and weighted tracks. We exclude MaxHS, which is described in Section 2.2.5.

- **UWrMaxSat**: It incrementally uses a MiniSat-like solver. In its main configuration, UWrMaxSat [177] applies an unsatisfiability-core-based OLL procedure [19, 168], and uses the kp-minisatp sorter-based pseudo-Boolean constraint encoding [116] to translate new cardinality constraints into CNF. It was developed on top of the PB-solver kp-minisatp, to get access to the pseudo-Boolean encoding with the same name.

- **RC2**: RC2 [106] is written in Python and based on the PySAT framework [105]. It is designed to serve as a simple example of how SAT-based problem solving algorithms can be implemented using PySAT while sacrificing just a little

Table 2.1: The MSE-2020 complete solvers classification and (un)weighted ranking.

Solver	Hitting Set	Unsat-based	Sat-Unsat	Unweighted	Weighted
EvalMaxSAT		✓		2	
MaxHS	✓			**1**	2
Pacose			✓		
QMaxSAT			✓		
UWrMaxSAT		✓	✓	3	**1**
maxino		✓			
Open-WBO		✓			
smax		✓			
RC2		✓			3

in terms of performance. In this sense, RC2 can be seen as a solver prototype and can be made somewhat more efficient if implemented in a low-level language. RC2 is written from scratch and implements the OLLITI (OLL-based) MaxSAT algorithm [168, 170], originally implemented in the MSCG MaxSAT solver [102, 170]. As PM2, WPM2 and WPM3, the RC2 solver deals with subproblems in the original instance. This is called core exhaustion in the RC2 context.

- **EvalMaxSAT**: EvalMaxSAT is based on the OLL algorithm [168] originally implemented in the MSCG MaxSAT solver [102, 170], and then reused in the RC2 solver [106]. The algorithm uses the mcqd library [119] to find a maximum clique in the incompatibility graph of soft clauses, and uses this clique to add at-most-one constraints between the soft clauses. The totalizer encoding [164] is used for cardinality constraints. The implementation reuses the code from the PySAT's ITotalizer [105]. EvalMaxSAT performs core minimization and core exhaustion, as exposed for RC2 [106].

All the solvers in the previous list are somehow based on the OLL procedure. They use soft cardinality constraints [168], core minimization and exhaustion. The UWrMaxSat solver outperforms MaxHS on the complete weighted track. MaxHS dominated the weighted track before appearing UWrMaxSat and RC2 in 2019.

2.2.7 Other contributions to MaxSAT solving

Other research topics in the literature that contain substantial contributions to the field of MaxSAT solving include the definition of clause learning schemes in BnB solvers [8, 43] that are not yet competitive enough to solve industrial instances, the creation of robust MaxSAT solutions [58], the definition of efficient encodings from MaxCSP to MaxSAT [32], the extension of MaxSAT to many-valued logic [34] and the definition of MaxSAT formalisms that deal with blocks of soft clauses instead of individual soft clauses [42, 96].

2.3 Complete MinSAT algorithms

Given the success of MaxSAT, the community has started to look into MinSAT. At first sight, one could think that the solving techniques and encodings to be used in MinSAT are very similar to the ones used in MaxSAT and, therefore, that there is no need of investigating MinSAT from a problem solving perspective. However, most of the research conducted so far indicates that they can be quite different, as well as that the performance profile of MaxSAT and MinSAT is different for several optimization problems represented in these formalisms [41, 103, 148]. Moreover, MinSAT is meaningful for both satisfiable and unsatisfiable instances, whereas MaxSAT is only meaningful for unsatisfiable instances. A closely related problem has been analyzed in [107, 108].

The work on MinSAT solving can be divided into the following categories:

- Transformations between MinSAT and MaxSAT: Reductions from MinSAT to Partial MaxSAT were defined in [139], but these reductions do not generalize to Weighted Partial MinSAT. This drawback was overcome with the definition of the natural encoding [123], which was improved in [203]. Reductions of Weighted Partial MinSAT to Group MaxSAT were evaluated in [97].

- Encodings from Weighted MaxCSP to MinSAT: Efficient encodings were defined in [40]. Using the MaxSAT direct encoding [32], we must add one clause for every no-good, while using the MinSAT direct encoding [40], we must instead add one clause for every good. This implies, for instance, that for representing the constraint $X = Y$, we need a number of clauses linear in the domain size in MinSAT, and a quadratic number of clauses in MaxSAT. We are in the opposite situation if we want to represent the constraint $X \neq Y$, So, it seems that MaxSAT and MinSAT could be complementary in some scenarios [40].

- Complete logical calculi: MaxSAT resolution is sound for MinSAT but the elimination of variables must be defined differently to get completeness [132]. After saturating a variable x, the clauses containing the variable x are ignored in MaxSAT. However, in MinSAT, the resulting clauses of eliminating the occurrences of both x and $\neg x$ must also be considered in the saturation of the next variable. In this way, after saturating all the variables, the number of empty clauses derived is equal to the maximum number of unsatisfied clauses.

- MinSAT solvers: The only existing BnB MinSAT solver, MinSatz [147, 148], is based on MaxSatz and implements upper bounds that exploit clique partition algorithms and MaxSAT technology. There exist two SAT-based MinSAT solvers [31, 97]. They differ with SAT-based MaxSAT solvers in the way of relaxing soft clauses. A local search MinSAT solver was described in [6].

Section 2.3.1 describes the BnB MinSAT solver MinSatz. We focus on this solver because it implements solving techniques that are not used in MaxSAT solving.

2.3.1 Branch-and-bound algorithms for MinSAT

We first present the MinSatz solver for Partial MinSAT, and explain how a good quality upper bound (UB) can be computed. We then extend the solver, and especially its UB, to Weighted Partial MinSAT.

MinSatz for Partial MinSAT

MinSatz implements the BnB scheme, and the search space is formed by a tree representing all the possible assignments. At every node, the solver starts by applying unit propagation using only hard unit clauses (i.e., given an existing or newly derived hard unit clause l, it satisfies and removes all the clauses containing the literal l, and removes all the occurrences of $\neg l$; soft unit clauses are not propagated because the simplified instance might have a different minimum number of satisfied clauses). If any hard clause becomes empty, then the solver backtracks. Otherwise, it computes an upper bound of the maximum number of soft clauses that will be unsatisfied (UB) if the current partial assignment is extended to a complete assignment. UB is then compared with the number of clauses unsatisfied by the best assignment found so far (LB), which is a lower bound. If UB$\leq LB$, a better solution cannot be found from the current node, and the solver backtracks. Otherwise, a variable is selected and instantiated. This process continues until all the search space has been explored, and the solver returns the best solution found. Algorithm 2.7 shows the pseudo-code of the MinSatz solver.

Algorithm 2.7: MinSatz(ϕ, LB)

```
  Input: $\phi$ , LB        ▷ A Partial MinSAT instance and the initial LB value (-1)

1  $\phi \leftarrow$ hardUnitPropagation($\phi$);
2  if $\phi$ contains a hard empty clause then
3      return LB;
4  if $\phi$={} or $\phi$ only contains empty clauses then
5      return $\max\{$#empty($\phi$), LB$\}$;
6  UB $\leftarrow$ #empty($\phi$)+overestimation($\phi$);
7  if (UB $\leq$ LB) then
8      return LB;
9  x $\leftarrow$ select($\phi$);
10 LB $\leftarrow$ MinSatz($\phi_x$, LB);
11 LB $\leftarrow$ MinSatz($\phi_{\neg x}$, LB);
12 return LB;
```

To solve an instance ϕ, we should call Algorithm 2.7 with the following parameters: MinSatz(ϕ, -1). If the algorithm returns -1, then the hard part of ϕ is unsatisfiable, and there is no feasible solution. Function #empty(ϕ) returns the number of empty soft clauses in ϕ, overestimation(ϕ) returns an overestimation of the maximum number of soft clauses that will be unsatisfied if the current partial assignment is extended to a complete assignment, and select(ϕ) implements the following variable selection heuristic: let $hard(l)$ ($soft(l)$) be the number of occurrences of literal l in hard (soft) clauses, and let $score(l) = 2 \times hard(l) + soft(l)$. Function select($\phi$) chooses a variable x with the highest value of $score(x) \times score(\neg x) + score(x) + score(\neg x)$. The instance ϕ_x ($\phi_{\neg x}$) is ϕ in which all clauses containing x ($\neg x$) are satisfied and removed, and the literal $\neg x$ (x) is removed from the remaining clauses. The MinSAT value for the input instance ϕ, i.e., the minimum number of satisfied clauses of ϕ is #soft(ϕ)-MinSatz(ϕ, -1), where #soft(ϕ) is the number of soft clauses in ϕ.

BnB MaxSAT solvers solve a MaxSAT instance by minimizing the number of unsatisfied clauses, while MinSatz solves a MinSAT instance by maximizing the number of unsatisfied clauses.

UB Computation for (Unweighted) Partial MinSAT

A decisive point to obtain fast MinSAT solvers is to equip them with good quality UBs. In this section, we describe the UB computation methods introduced in [148], which are based on first computing a clique partition in a graph that is built from the current MinSAT instance, and then improving the obtained UB with MaxSAT technology. We first describe how UB is computed in the unweighted case and then describe three different methods for computing UB in the weighted case.

Assume that we are in a node of the search space and, after applying unit propagation using only hard unit clauses, we have an instance ϕ formed by the hard clauses $\{h_1, \ldots, h_k\}$, e empty soft clauses, and the not yet decided soft clauses $\{c_1, \ldots, c_m\}$. We start by building an undirected graph $G = (V, E)$, where V contains an element for every soft clause in $\{c_1, \ldots, c_m\}$, say $V = \{v_1, \ldots, v_m\}$. We add an edge between vertex v_i, corresponding to clause $c_i = \{l_1^i, \ldots, l_p^i\}$, and vertex v_j, corresponding to clause $c_j = \{l_1^j, \ldots, l_q^j\}$, if there exist a literal l_a^i and a literal l_b^j such that l_a^i=$\neg l_b^j$, or the set of clauses $\{\neg l_1^i, \ldots, \neg l_p^i, \neg l_1^j, \ldots, \neg l_q^j, h_1, \ldots, h_k\}$ may be declared to be unsatisfiable using unit propagation. The idea behind the graph G, called the graph associated with ϕ, is that the clauses associated with the two vertices of an edge cannot be simultaneously unsatisfied. Indeed, in the former case, two literals in c_i and c_j are complementary and, in the latter case, if c_i and c_j are both unsatisfied, then a hard clause is violated.

Once the graph G is built, an overestimation of the maximum number of soft clauses that can be unsatisfied corresponds to a Maximum Independent Set (MIS) of G, where there exists no edge between any two vertices of the independent set. We could then compute an UB of the maximum number of soft clauses that can be unsatisfied by computing an UB of the cardinality of a MIS of G. For this purpose, we create a clique partition of G using the heuristic algorithm described in [187], and also used in [145, 146]: Suppose that the vertices are sorted by increasing order of their degree and that the current partition is $S_1, S_2, \ldots, S_k$ (in this order, and being $k = 0$ at the beginning). The algorithm inserts the first vertex v of the sequence of vertices into the first S_i such that v is connected to all the vertices already in S_i. If such S_i does not exist, a new set S_{k+1} is created and v is inserted in S_{k+1}. This process is repeated until there are no more vertices left.

By construction of graph G, there is at most one unsatisfied clause in every clique. In other words, at least all the clauses in a clique except one are satisfied by any complete truth assignment satisfying all the hard clauses. Hence, the number of cliques in the partition, say s, is an overestimation of the number of clauses that can be unsatisfied if the current partial assignment is completed. Taking into account this fact, we define UB= $e + s$. However, UB might not be tight enough. if G is not perfect, then UB is not tight because the number of cliques in the partition of G is an upper bound of the chromatic number $\chi(\bar{G})$ of the complementary graph of G ($\bar{G}$), which is strictly larger than the cardinality of a MIS of G. If G is perfect, then UB can be tight but it is not guaranteed.

In order to improve UB, an approach adapted from [145, 146] can be used. We derive a Partial MaxSAT instance ψ from the obtained clique partition of graph G: for every edge (v_i, v_j) of the graph, we add the hard clause $\neg v_i \vee \neg v_j$ and, for every clique $\{v_{i_1}, \ldots, v_{i_k}\}$, we add the soft clause $v_{i_1} \vee \cdots \vee v_{i_k}$. Hard clauses in ψ state that clauses of the original MinSAT instance ϕ associated with adjacent vertices of G cannot be simultaneously unsatisfied, while soft clauses in ψ state that at least one soft clause of ϕ associated with the vertices of a clique of G is unsatisfied. It turns out that if an optimal solution of the resulting Partial MaxSAT instance ψ has u unsatisfied soft clauses, then we can decrement UB by u, because we can conclude that u cliques cannot contain any unsatisfied soft clause in ϕ and should not be counted in the UB of the number of unsatisfied soft clauses in ϕ. Observe that v_{i_k} is true if clause c_{i_k} in ϕ is unsatisfied, and false if it is satisfied. Hence, if a soft clause $v_{i_1} \vee \cdots \vee v_{i_k}$ is violated, all the clauses of ϕ associated with the clique $\{v_{i_1}, \ldots, v_{i_k}\}$ are satisfied; in other words, that clique does not contain any unsatisfied clause.

Since solving Partial MaxSAT is NP-hard, in practice, u is underestimated by using the technology developed for MaxSAT solvers. More precisely, lower bound UP enhanced with failed literal detection [136, 137] is applied to the derived Partial MaxSAT instance ψ. In this MinSAT setting, UB is improved by decrementing the total number of contradictions detected with UP enhanced with failed literal detection.

Example 2.1. Assume that we are in a node in which we have the hard clauses $\neg x_1 \vee \neg x_2$, $\neg x_2 \vee \neg x_3$, $\neg x_3 \vee \neg x_4$, $\neg x_4 \vee \neg x_5$, $\neg x_1 \vee \neg x_5$, and the soft clauses $\neg x_1, \neg x_2, \neg x_3, \neg x_4, \neg x_5$. Also assume that no clause has yet become empty. We build the graph G associated with the instance, which is shown in Figure 2.1. The set of vertices is $\{v_1, v_2, v_3, v_4, v_5\}$, where vertex v_i is associated with the soft clause $\neg x_i$, for $1 \leq i \leq 5$. The set of edges is $\{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_5), (v_1, v_5)\}$. Assume that the algorithm finds the following clique partition of G: $\{\{v_1, v_2\}, \{v_3, v_4\}, \{v_5\}\}$. Then, at most 3 soft clauses among 5 soft clauses can be unsatisfied, and UB=3.

G is not perfect and, therefore, UB is not tight. A deeper analysis shows that only

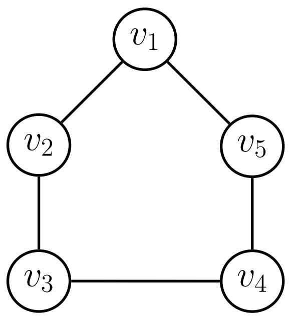


Figure 2.1: Graph associated with the MinSAT instance of Example 2.1

two clauses of the instance can be unsatisfied (instead of 3) so that UB might be improved to 2.

Assume that every clique contains an unsatisfied clause under some complete assignment. Then, v_5 should be unsatisfied, but v_1 and v_4 cannot be unsatisfied because v_1 and v_4 are connected to v_5. So, the only possibility for the first and the second cliques to have an unsatisfied clause is that v_2 and v_3 are both unsatisfied, but this is not possible because v_2 and v_3 are connected. Hence, $\{\{v_1, v_2\}, \{v_3, v_4\}, \{v_5\}\}$ is a subset of cliques in which not all cliques can have an unsatisfied clause. Therefore, we could decrement UB by one. In order to detect such a situation, we derive the Partial MaxSAT instance formed by the hard clauses:

$$\begin{array}{lll} \neg v_1 \vee \neg v_2 & \neg v_1 \vee \neg v_5 & \neg v_2 \vee \neg v_3 \\ \neg v_3 \vee \neg v_4 & \neg v_4 \vee \neg v_5 & \end{array}$$

and the soft clauses:

$$v_1 \vee v_2 \qquad v_3 \vee v_4 \qquad v_5$$

Then, we apply UP enhanced with failed literal detection. Actually, we detect a contradiction applying just UP to the above instance. It corresponds to the refutation that can be constructed from $\{\neg v_1 \vee \neg v_5, \neg v_2 \vee \neg v_3, \neg v_4 \vee \neg v_5, v_1 \vee v_2, v_3 \vee v_4, v_5\}$. Since all the soft clauses are involved in the refutation, we cannot derive additional contradictions. Therefore, we decrement UB by one and get UB=2.

Extending MinSatz and its UB to Weighted Partial MinSAT

In the weighted case, the objective is to find an assignment that satisfies all the hard clauses and minimizes the sum of weights of satisfied soft clauses. To reach this objective, MinSatz finds an assignment that satisfies all the hard clauses and maximizes the sum of weights of unsatisfied soft clauses. Now, in Algorithm 2.7, #empty(ϕ) returns the sum of weights of all the empty soft clauses in ϕ, overestimation(ϕ) returns an overestimation of the maximum sum of weights of the soft clauses that will be unsatisfied if the current partial assignment is completed, soft(ϕ) is the total sum of the weights of the soft clauses in ϕ, and the MinSAT value for the input instance ϕ is $soft(\phi) - MinSat(\phi, -1)$.

The variable selection heuristic works as follows in the weighted case: Let $hard(l)$ be the number of occurrences of literal l in hard clauses, let $soft(l)$ be the sum of weights of the soft clauses containing literal l, let $weight(l)$ be the mean of the weights among all the soft clauses containing literal l, and let the score for the literal l be $score(l) = 2 \times weight(l) \times hard(l) + soft(l)$. Select($\phi$) chooses a variable maximizing $score(x) \times score(\neg x) \times (weight(x) \times weight(\neg x)/2) + score(x) + score(\neg x)$.

In the rest of the section, we explain how the computation method described above for UB can be extended to the weighted case. We obtain three different methods and refer to them as UB1, UB2 and UB3. In all of them, the graph associated with the MinSAT instance is built as in the unweighted case. The difference lies in that now the graph is weighted, and in how the clique partition is created and the weights are operated.

We define the weight of a vertex of G as the weight of the corresponding soft clause, and the weight w of a clique $\{v_{i_1}, \ldots, v_{i_n}\}$ as the minimum weight among the weighted soft clauses $(c_{i_1}, w_{i_1}), \ldots, (c_{i_n}, w_{i_n})$ (i.e., $w = \min(w_{i_1}, \ldots, w_{i_n})$).

- **Upper Bound UB1**

  UB1 creates a clique partition in the graph G associated with the current MinSAT instance using the same heuristic algorithm as in the unweighted case. Let (v_i, w_i) be a vertex in a clique of weight w. Then, it constructs a subgraph G' induced by those vertices v_i of G such that $w_i - w > 0$, and defines the weight of v_i in G' as $w_i - w$. G' is in turn partitioned into cliques, and a subgraph of G' is constructed in the same way for finding further partitions. This process, whose pseudo-code is shown in Algorithm 2.8, continues until the empty graph is obtained. Finally, UB1 is computed taking into account the weights of all the obtained cliques: UB1=$\sum_{clause\ c_i\ is\ empty} w_i + \sum_{all\ cliques} w$.

Algorithm 2.8: Partition(ϕ)

```
Input: $\phi$                                   ▷ A Partial MinSAT instance

1 Construct a weighted graph $G$ from $\phi$;
2 $P \leftarrow \{\}$;
3 repeat
4     Find a clique partition of $G$, and add the cliques into $P$;
5     Construct $G$' from the cliques and $G$;
6     $G \leftarrow G$';
7 until $G$ becomes empty;
8 return $P$;
```

Algorithm 2.8 returns a partition of G into cliques, in such a way that all vertices in each clique have the same weight in order to compute a tight upper bound

UB1. Observe that there are many ways to partition a weighted graph into cliques in which all vertices have the same weight. Some of them could give a huge number of cliques, especially when the vertex weights are very different and the graph is large. The approach presented in Algorithm 2.8 does not necessarily give the best upper bound. However, it guarantees that the number of vertices in G' is at most $|V| - s$, where V is the set of vertices of G and s is the number of cliques in its clique partition, because at least one vertex in each clique has the same weight as the clique before the subtraction and cannot belong to G', which seems a reasonable approach in the implementation of a MinSAT solver.

As in the unweighted case, UP enhanced with failed literal detection is applied to improve the upper bound. Now, for every clique $\{v_{i_1}, \ldots, v_{i_k}\}$, we add the weighted soft clause $(v_{i_1} \vee \cdots \vee v_{i_k}, w)$, being w the weight of the clique. Every time we detect a subset of cliques in which not all cliques can have unsatisfied clauses, we improve the upper bound by w, where w is the minimum weight among all the cliques in the subset.

Example 2.2. Consider the MinSAT instance formed by the hard clauses $\neg x_1 \vee \neg x_2$, $\neg x_2 \vee \neg x_3$, $\neg x_3 \vee \neg x_4$, $\neg x_4 \vee \neg x_5$, $\neg x_1 \vee \neg x_5$, and the soft clauses ($\neg x_1$, 2), ($\neg x_2$, 3), ($\neg x_3$, 4), ($\neg x_4$, 5), ($\neg x_5$, 6). The graph associated with that instance is shown in Figure 2.1. Algorithm 2.8 returns a weighted clique partition {{(v_1, v_2), 2}, {(v_3, v_4), 4}, {(v_5), 6}, {(v_2), 1}, {(v_4), 1}}, getting UB1=14. To improve UB1, UP enhanced with failed literal detection is applied to the following Weighted Partial MaxSAT instance:

$$\begin{array}{lll}
\neg v_1 \vee \neg v_2 & \neg v_1 \vee \neg v_5 & \neg v_2 \vee \neg v_3 \\
\neg v_3 \vee \neg v_4 & \neg v_4 \vee \neg v_5 & \\
(v_1 \vee v_2, 2) & (v_3 \vee v_4, 4) & (v_2, 1) \\
(v_4, 1) & (v_5, 6) &
\end{array}$$

As a result, the unsatisfiable subset $\neg v_1 \vee \neg v_5, \neg v_2 \vee \neg v_3, \neg v_4 \vee \neg v_5, (v_1 \vee v_2, 2), (v_3 \vee v_4, 4), (v_5, 6)$ is detected. We decrement UB1 by 2 because it is the minimum among 2, 4, and 6. Therefore, UB1=12.

Notice that, depending on the order in which unit clauses were selected by UP enhanced with failed literal detection, UB1 could instead detect the unsatisfiable subset $\neg v_4 \vee \neg v_5, (v_4, 1), (v_5, 6)$. In this case, we could decrement UB1 just by 1.

- **Upper Bound UB2**

UB2 is an upper bound that builds a clique partition in an incremental way: it starts by selecting a vertex v with minimum degree, breaking ties by selecting a vertex with minimum weight. Then, it computes a maximal clique containing v, and decrements the weight of every vertex of G in the computed clique by the weight of the clique. It removes the vertices with weight zero (so, it removes at least one vertex), and repeats this process until the empty graph is derived.

The sum of the weights of all the computed cliques is UB2. Finally, UB2 is improved by applying UP enhanced with failed literal detection to a Weighted Partial MaxSAT instance as in UB1.

Example 2.3.

We consider the same MinSAT instance as in Example 2.2. In the first step, UB2 selects vertex v_1 and finds the clique $\{v_1, v_2\}$ with weight 2. Then, it removes vertex v_1 from the graph and the weight of vertex v_2 is set to 1. In the second step, UB2 selects vertex v_2 and finds the clique $\{v_2, v_3\}$ with weight 1. Then, it removes vertex v_2 from the graph and the weight of vertex v_3 is set to 3. In the third step, UB2 selects vertex v_3 and finds the clique $\{v_3, v_4\}$ with weight 3. Then, it removes vertex v_3 from the graph and the weight of vertex v_4 is set to 2. In the fourth step, UB2 selects vertex v_4 and finds the clique $\{v_4, v_5\}$ with weight 2. Then, it removes vertex v_4 from the graph and the weight of vertex v_5 is set to 4. In the fifth step, UB2 selects vertex v_5 and finds the clique $\{v_5\}$ with weight 4. Then, it removes vertex v_5, gets the empty graph and UB2=12.

To improve UB2, UP enhanced with failed literal detection is applied to the following Weighted Partial MaxSAT instance:

$$\begin{array}{lll}
\neg v_1 \vee \neg v_2 & \neg v_1 \vee \neg v_5 & \neg v_2 \vee \neg v_3 \\
\neg v_3 \vee \neg v_4 & \neg v_4 \vee \neg v_5 & \\
(v_1 \vee v_2, 2) & (v_2 \vee v_3, 1) & (v_3 \vee v_4, 3) \\
(v_4 \vee v_5, 2) & (v_5, 4) &
\end{array}$$

As a result, it is detected that the soft clauses $(v_1 \vee v_2, 2), (v_3 \vee v_4, 3), (v_5, 4)$, when added to the hard clauses, are unsatisfiable. We decrement UB2 by 2 because it is the minimum among 2, 3, and 4. Therefore, UB2=10. Notice that UB2 is tighter than UB1 for the analyzed MinSAT instance. When we remove the soft clauses $(v_1 \vee v_2, 2), (v_3 \vee v_4, 3), (v_5, 4)$, we cannot detect any other contradiction. Actually, 10 is the optimal UB value for the present MinSAT instance.

It is worth pointing out that UB2 applied to an unweighted graph is slower than UB1 because UB2 computes a clique at a time while UB1 can compute several cliques at a time.

- **Upper Bound UB3**

  UB3 can be seen as an upper bound that improves UB2. It incrementally computes a clique partition as UB2, and the difference lies in the fact that, in the subsets of soft clauses detected by UP enhanced with failed literal detection, the weights of cliques are taken into account, in the sense that the cliques with weight greater than the minimum weight can be used to detect additional contradictions. Of course, we must update the weights of cliques after detecting a contradiction. We illustrate this improvement using another MinSAT instance because UB2 is optimal for the instance in Example 2.2.

Example 2.4.

Consider the MinSAT instance formed by the hard clauses $\neg x_1 \vee \neg x_2$, $\neg x_2 \vee \neg x_3$, $\neg x_3 \vee \neg x_4$, $\neg x_4 \vee \neg x_5$, $\neg x_1 \vee \neg x_5$, $\neg x_5 \vee \neg x_6$, $\neg x_6 \vee \neg x_7$, $\neg x_7 \vee \neg x_8$, $\neg x_8 \vee \neg x_9$, $\neg x_5 \vee \neg x_9$ and the soft clauses ($\neg x_1$, 2), ($\neg x_2$, 3), ($\neg x_3$, 4), ($\neg x_4$, 5), ($\neg x_5$, 9), ($\neg x_6$, 3), ($\neg x_7$,1), ($\neg x_8$, 3), ($\neg x_9$, 3). The graph associated with that instance is shown in Figure 2.2.

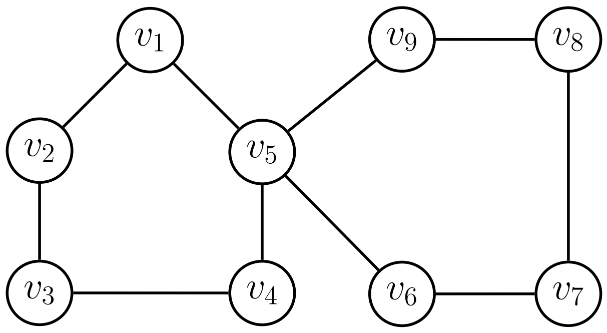


Figure 2.2: Graph associated with the MinSAT instance of Example 2.4

In the first step, UB3 selects vertex v_7 and finds the clique $\{v_6, v_7\}$ with weight 1. Then, it removes vertex v_7 from the graph and the weight of vertex v_6 is set to 2. In the second step, UB3 selects vertex v_6 and finds the clique $\{v_5, v_6\}$ with weight 2. Then, it removes vertex v_6 from the graph and the weight of vertex v_5 is set to 7. In the third step, UB3 selects vertex v_8 and finds the clique $\{v_8, v_9\}$ with weight 3. Then, it removes vertices v_8 and v_9. In the fourth step, UB3 selects vertex v_1 and finds the clique $\{v_1, v_2\}$ with weight 2. Then, it removes vertex v_1 from the graph and the weight of vertex v_2 is set to 1. In the fifth step, UB3 selects vertex v_2 and finds the clique $\{v_2, v_3\}$ with weight 1. Then, it removes vertex v_2 from the graph and the weight of vertex v_3 is set to 3. In the sixth step, UB3 selects vertex v_3 and finds the clique $\{v_3, v_4\}$ with weight 3. Then, it removes vertex v_3 from the graph and the weight of vertex v_4 is set to 2. In the seventh step, UB3 selects vertex v_4 and finds the clique $\{v_4, v_5\}$ with weight 2. Then, it removes vertex v_4 from the graph and the weight of vertex v_5 is set to 5. In the eighth step, UB3 selects vertex v_5 and finds the clique $\{v_5\}$ with weight 5. Then, it removes vertex v_5, gets the empty graph and UB3=19.

To improve UB3, UP enhanced with failed literal detection is applied to the following Weighted Partial MaxSAT instance:

$$\begin{array}{lll}
\neg v_1 \vee \neg v_2 & \neg v_1 \vee \neg v_5 & \neg v_2 \vee \neg v_3 \\
\neg v_3 \vee \neg v_4 & \neg v_4 \vee \neg v_5 & \neg v_5 \vee \neg v_6 \\
\neg v_5 \vee \neg v_9 & \neg v_6 \vee \neg v_7 & \neg v_7 \vee \neg v_8 \\
\neg v_8 \vee \neg v_9 & & \\
(v_1 \vee v_2, 2) & (v_2 \vee v_3, 1) & (v_3 \vee v_4, 3) \\
(v_4 \vee v_5, 2) & (v_5, 5) & (v_5 \vee v_6, 2) \\
(v_6 \vee v_7, 1) & (v_8 \vee v_9, 3) &
\end{array}$$

UB3 detects that the soft clauses $(v_1 \vee v_2, 2), (v_3 \vee v_4, 3), (v_5, 5)$, when added to the hard clauses, are unsatisfiable. Now, it removes $(v_1 \vee v_2, 2)$, and replaces $(v_3 \vee v_4, 3)$ with $(v_3 \vee v_4, 1)$ and replaces $(v_5, 5)$ with $(v_5, 3)$. Now, UB3 detects that the soft clauses $(v_6 \vee v_7, 1), (v_8 \vee v_9, 3), (v_5, 3)$, when added to the hard clauses, are unsatisfiable. It removes $(v_6 \vee v_7, 1)$, and replaces $(v_8 \vee v_9, 3)$ with $(v_8 \vee v_9, 2)$ and replaces $(v_5, 3)$ with $(v_5, 2)$. Since no additional contradictions can be detected, $UB3 = 16$. Notice that $UB2 = 17$ for this instance because after detecting the first contradiction the unit soft clause containing v_5 is removed. UB2 does not update the weights as UB3.

2.4 Concluding remarks

We have defined the MaxSAT and MinSAT problems and presented the most relevant algorithms and techniques used to solve them. We have focused on exact MaxSAT and MinSAT solving for two reasons: (i) all the contributions of this thesis were created for exact solvers, and (ii) the research in the SAT community has mainly focused on exact MaxSAT and MinSAT solving in the last decade.

Chapter 3

Clausal Tableau Calculi for MaxSAT and MinSAT

This chapter defines a tableau calculus for solving MaxSAT, a tableau calculus for solving MinSAT, and a tableau calculus for solving both MaxSAT and MinSAT. For each calculus, a proof of soundness and completeness is provided. It holds that the minimum number of contradictions derived among the branches of a completed MaxSAT tableau for a multiset of clauses ϕ is the minimum number of unsatisfied clauses in ϕ, and the maximum number of contradictions derived among the branches of a completed MinSAT tableau for a multiset of clauses ϕ' is the maximum number of unsatisfied clauses in ϕ'. It also describes how the new calculi can be extended to deal with Weighted Partial MaxSAT and MinSAT instances.

The presentation closely follows the paper published in the Logic Journal of the IGPL [39]. It unifies results on clausal MaxSAT tableaux published in [140], results on clausal MinSAT tableaux published in [141], and results on a tableau-based procedure for MaxSAT and MinSAT published in [38].

3.1 Introduction

The current inference systems for SAT are unsound for solving MaxSAT and MinSAT. They preserve satisfiability but do not preserve the minimum and/or the maximum number of unsatisfied clauses. Thus, to define complete inference systems for MaxSAT and MinSAT, we first need to define cost-preserving inference rules and then show that their correct application allows one to derive as many empty clauses as the minimum number of unsatisfied clauses in the case of MaxSAT, and as the maximum number of unsatisfied clauses in the case of MinSAT.

Resolution-style procedures, based on the MaxSAT resolution rule have been defined for MaxSAT [63, 64] and MinSAT [132]. The difference between MaxSAT and MinSAT lies in the way of eliminating variables. Moreover, refinements of the MaxSAT resolution rule have been incorporated into BnB MaxSAT and MinSAT algorithms and have produced important speedups (see e.g. [4, 7, 124, 135, 138, 148]).

Even when resolution-based methods are competitive for solving clausal MaxSAT and MinSAT, they present some limitations when dealing with more expressive formalisms. Thus, we decided to open a new research direction in MaxSAT and MinSAT solving: the definition of complete tableau-like calculi for MaxSAT and MinSAT.

The advantages of tableaux over resolution are that tableaux allow one to solve directly non-clausal MaxSAT and MinSAT, can be naturally extended to first-order logic and are more appropriate for dealing with non-classical logics in which it is not easy to define clausal forms.

In this chapter, we first extend the clause tableau calculus for SAT [73, 92] to solve both MaxSAT [140] and MinSAT [141]. An essential issue to address is how to derive simpler subproblems so that the minimum/maximum number of unsatisfied clauses is preserved. The proposed clause MaxSAT tableau calculus and the clause SAT tableau calculus are quite similar, in that they use the same rules but applied differently. The clause MinSAT tableau calculus has an inference rule that does not resemble the rules of the SAT and MaxSAT calculi. It is worth noticing that a clause MinSAT tableau calculus must be able to derive contradictions from satisfiable instances because they can have interpretations that falsify some clauses.

From the insights gained after analyzing clause MaxSAT and MinSAT tableaux, we propose a clause tableau calculus that is valid for both MaxSAT and MinSAT. It adequately preserves the number of unsatisfied clauses in the generated subproblems. The leaf nodes of a completed search tree contain a number of empty clauses ranging between the minimum and the maximum number of unsatisfied clauses in the input formula, and there is at least one branch with the minimum value and at least one branch with the maximum value. This calculus also provides optimal MaxSAT and MinSAT assignments by inspecting the optimal branches.

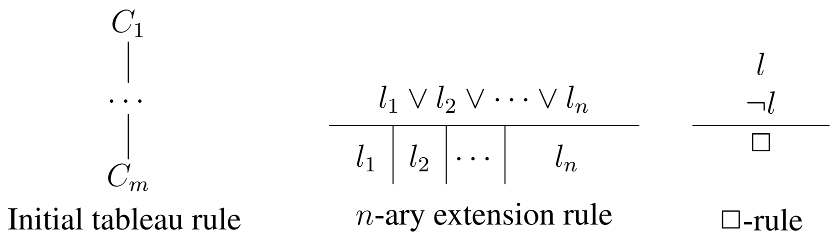
For the sake of clarity, we first present the tableau calculi for Unweighted MaxSAT and MinSAT. We then explain how the proposed calculi can be extended to solve both Weighted Partial MaxSAT and Weighted Partial MinSAT.

The rest of the chapter is organized as follows. Section 3.2 reviews how clause tableaux are used to solve SAT. Section 3.3 defines a clause tableau calculus for MaxSAT and proves its completeness. Section 3.4 defines a clause tableau calculus for MinSAT and proves its completeness. Section 3.5 describes a clause tableau calculus that is valid for both MaxSAT and MinSAT and proves its completeness. Section 3.6 concludes the chapter.

3.2 Clause tableaux for SAT

It is common to view the tableau method for solving SAT as a proof by case distinction that allows one to systematically generate subcases until elementary contradictions are reached [85, 186]. In the context of SAT, a clause tableau is a tree with a finite number of branches whose nodes are labelled with clauses, and a branch is a maximal path in a tree with a finite number of nodes. A branch is closed if two nodes are labelled with complementary unit clauses; otherwise, it is open. A clause tableau is closed iff all its

Table 3.1: Expansion rules of a complete clause tableau calculus for SAT



branches are closed.

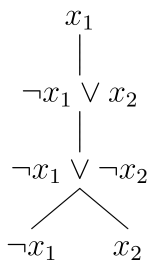
Given a set of clauses $\phi = \{C_1, \ldots, C_m\}$, we start by creating an initial tableau with a single branch of m nodes, where each node is labelled with a clause of ϕ. This process is known as the application of the initial tableau rule. Then, we select an open branch B and a clause $l_1 \vee \ldots \vee l_r$ of ϕ with $r \geq 2$ that has not yet been expanded in B, and append r sibling nodes below B, labelling each node with a different unit clause from $\{l_1, \ldots, l_r\}$. This process of creating r new branches from B is known as the application of the extension rule. If there are two complementary unit clauses in a branch, we close it by applying the contradiction rule. In the following, closing a branch amounts to deriving an empty clause. This process continues until either all the branches are closed, or the application of the extension rule on a branch until saturation leaves it open. The set of clauses ϕ is declared to be unsatisfiable in the first case, and satisfiable in the second case. Table 3.1 shows the expansion rules of a complete clause tableau calculus for SAT.

Example 3.1. To determine the satisfiability of $\phi = \{x_1, \neg x_1 \vee x_2, \neg x_1 \vee \neg x_2\}$ with clause tableaux we start by creating the initial tableau (T_0):

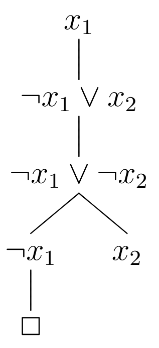
$$\begin{tikzcd}
x_1 \arrow[d, dash] \\
\neg x_1 \vee x_2 \arrow[d, dash] \\
\neg x_1 \vee \neg x_2
\end{tikzcd}$$

(T_0)

We then expand the second node (T_1) and close the leftmost branch by applying the contradiction rule to x_1 and $\neg x_1$, obtaining another clause tableau (T_2):

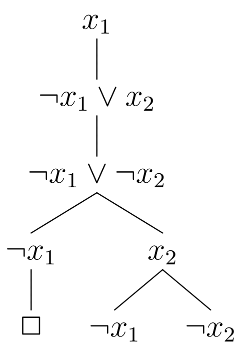


(T_1)

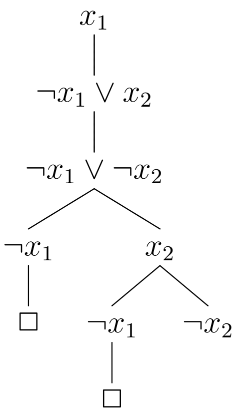


(T_2)

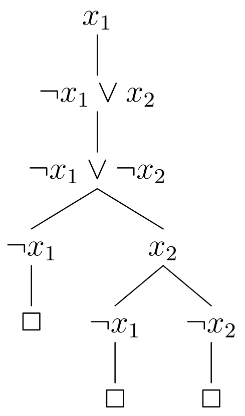
Finally, we expand the third node on the rightmost branch (T_3), and close the new leftmost (T_4) and rightmost branches (T_5), obtaining a clause tableau proof of the unsatisfiability of ϕ:



(T_3)



(T_4)



(T_5)

Formally, a clause tableau proof of the unsatisfiability of a set of clauses ϕ is a sequence of clause tableaux $T_0, \ldots, T_n$ such that T_0 is an initial tableau of ϕ, T_n is a closed tableau, and T_i has been obtained by a single application of the extension or contradiction rule on an open branch of T_{i-1} for $i = 1, \ldots, n$. In a proof of satisfiability, T_n must have some open branch after applying the extension rule on it until saturation. Besides, the literals occurring in the unit clauses of the open branch provide a satisfying assignment of ϕ. It is common to say that T_n is a clause tableau proof because it collapses all the sequence of tableaux. In Example 3.1, the sequence $T_0, T_1, T_2, T_3, T_4, T_5$ is a clause tableau proof, although T_5 alone is also considered to be a proof.

We say that a clause tableau is completed when all its branches are closed or it contains an open branch in which it was not possible to detect a contradiction with the expansion rules of Table 3.1.

From a semantic perspective, given a set of clauses ϕ and the completed clause tableau T for ϕ, we have that ϕ is satisfiable iff there is a branch of T such that the conjunction of all its literals is satisfiable. Alternatively, ϕ is unsatisfiable iff all the branches of T are unsatisfiable.

3.3 Clause tableaux for MaxSAT

The clause SAT tableau calculus is not valid for solving MaxSAT, but it can become sound and complete by applying the expansion rules of Table 3.1 differently.

Firstly, the application of expansion rules in a branch cannot stop once a contradiction is detected. Since MaxSAT aims to derive all the possible contradictions, the application of rules in a branch should continue until no more expansion rules can be applied. Thus, a different notion of completed tableau is needed.

Secondly, the application of rules in SAT leads to accumulating the newly added unit clauses in the branch so that satisfiability is preserved in at least one branch when the input set of clauses is satisfiable. However, the addition of redundant clauses might lead to wrong MaxSAT solutions. In clause MaxSAT tableaux, the goal should be to keep the minimum number of unsatisfied clauses in at least one branch and not decrease that number in the rest of branches. As we show below, the rules of Table 3.1 satisfy that condition provided that we maintain active and inactive clauses. In other words, once a clause has been used as a premise of a rule in a branch, it cannot be used again in that branch and becomes inactive. For example, thanks to distinguishing between active and inactive clauses, we will detect one contradiction in the multiset of unit clauses $\{x_1, \neg x_1, \neg x_1\}$ and two in $\{x_1, x_1, \neg x_1, \neg x_1\}$. Without that, we could detect two contradictions in the first case, obtaining a wrong answer. In fact, the inference rules of MaxSAT can be seen as rewriting rules.

Definition 3.1. A clause MaxSAT tableau is a tree with a finite number of branches whose nodes are labelled with clauses. A branch is a maximal path in a tree, and we assume that branches have a finite number of nodes.

Definition 3.2. Let $\phi = \{C_1, \ldots, C_m\}$ be a multiset of clauses. A clause MaxSAT tableau for ϕ is constructed by a sequence of applications of the following expansion rules:

Initialize A tree with a single branch with m nodes such that each node is labelled with a clause of ϕ is a clause MaxSAT tableau for ϕ. Such a tableau is called initial tableau and its clauses are declared to be active.

Extension Given a clause MaxSAT tableau T for ϕ, a branch B of T, and a node of B labelled with an active clause $l_1 \vee \cdots \vee l_r$ with $r \geq 2$, the tableau obtained by creating r sibling nodes below B and labelling each node with a different unit clause from $\{l_1, \ldots, l_r\}$ is a clause MaxSAT tableau for ϕ. The clause $l_1 \vee \cdots \vee l_r$ becomes inactive in the new branches, and the unit clauses $l_1, \ldots, l_r$ are declared to be active.

Contradiction Given a clause MaxSAT tableau T for ϕ, a branch B of T, and two nodes of B labelled with two active unit clauses l and $\neg l$, the tableau obtained by appending a node labelled with an empty clause ($\Box$) below B is a clause MaxSAT tableau for ϕ. The unit clauses l and $\neg l$ become inactive in B and the added empty clauses ($\Box$) is considered active.

Definition 3.3. Let T be a clause MaxSAT tableau for a multiset of clauses ϕ, and let B be a branch of T. Branch B is saturated when all its active clauses are unit or empty, and the contradiction rule cannot be further applied on B. Tableau T is completed iff all its branches are saturated. The cost of a saturated branch is the number of empty clauses in it. The cost of a completed clause MaxSAT tableau is the minimum cost among all its branches.

The notion of saturation is crucial in MaxSAT because it indicates that the application of expansion rules has been completed. As we show below, the minimum number of clauses that can be falsified in a multiset of clauses ϕ is k iff the cost of any completed clause MaxSAT tableau for ϕ is k. Hence, the systematic construction of a completed clause MaxSAT tableau for ϕ provides an exact method for MaxSAT, and each completed tableau is a proof.

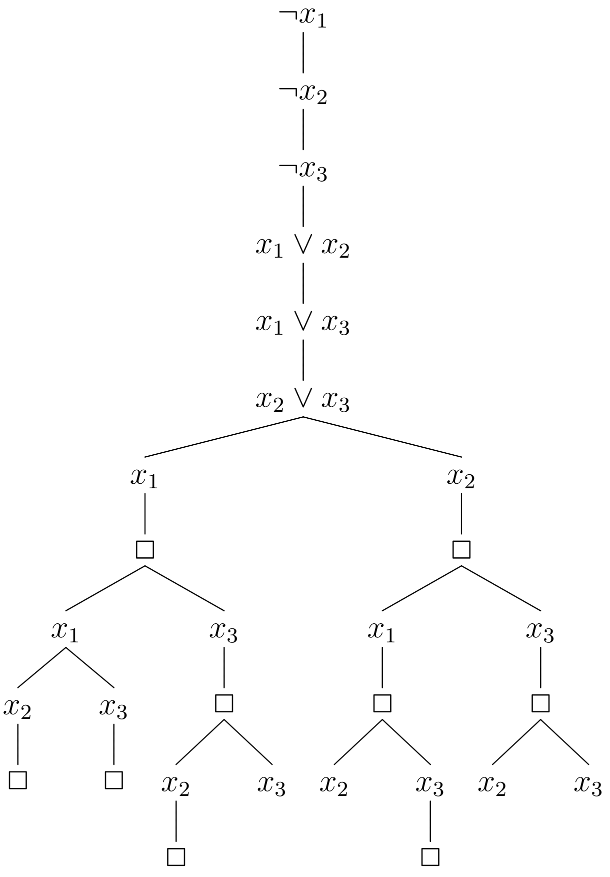


Figure 3.1: A completed clause MaxSAT tableau for $\phi = \{\neg x_1, \neg x_2, \neg x_3, x_1 \vee x_2, x_1 \vee x_3, x_2 \vee x_3\}$ that proves that the minimum number of unsatisfied clauses in ϕ is 2.

Example 3.2. Let $\phi = \{\neg x_1, \neg x_2, \neg x_3, x_1 \vee x_2, x_1 \vee x_3, x_2 \vee x_3\}$ be a multiset of clauses. Figure 3.1 shows a completed clause MaxSAT tableau T for ϕ, and Figure 3.2 shows the steps performed for saturating the leftmost branch of T.

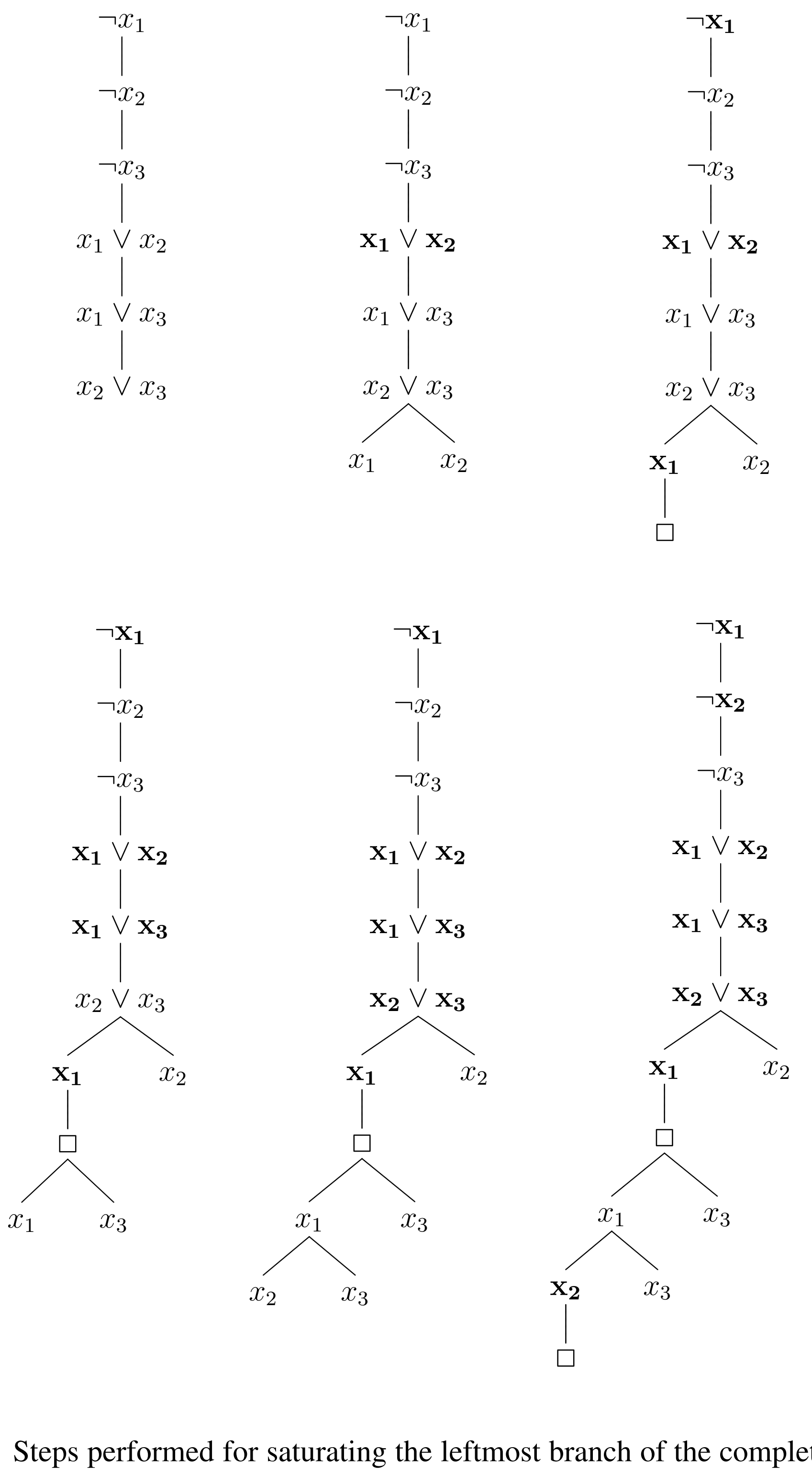


Figure 3.2: Steps performed for saturating the leftmost branch of the completed clause MaxSAT tableau for $\phi = \{\neg x_1, \neg x_2, \neg x_3, x_1 \vee x_2, x_1 \vee x_3, x_2 \vee x_3\}$ from Example 3.2.

In Figure 3.2, we first create an initial tableau. Secondly, we apply the extension rule to clause $x_1 \vee x_2$, and declare it inactive in the newly created branches (in the figure we write in bold the inactive clauses in the leftmost branch, which is the branch on which we concentrate in this example). Thirdly, we apply the contradiction rule to $\neg x_1$ and x_1, and declare these clauses inactive in the leftmost branch. Fourthly, we apply the extension rule to $x_1 \vee x_3$, and declare it inactive in the newly created branches. Fifthly, we apply the extension rule to $x_2 \vee x_3$, and declare it inactive. Sixthly, we apply the contradiction rule to $\neg x_2$ and x_2, and declare these clauses inactive in the leftmost branch. No more inference rules can be applied on the leftmost branch and therefore the branch is saturated, having as active clauses $\{\Box, \Box, x_1, \neg x_3\}$. A similar process is repeated to create the rest of branches in Figure 3.1.

The saturated branches of the tableau of Figure 3.1 have cost 2 except for branches 3 and 6 (counting from left to right) that have cost 3. The active clauses in each branch are: $\{\Box, \Box, x_1, \neg x_3\}$ for the first branch, $\{\Box, \Box, x_1, \neg x_2\}$ for the second, $\{\Box, \Box, \Box\}$ for the third, $\{\Box, \Box, \neg x_2, x_3\}$ for the fourth, $\{\Box, \Box, x_2, \neg x_3\}$ for the fifth, $\{\Box, \Box, \Box\}$ for the sixth, $\{\Box, \Box, \neg x_1, x_2\}$ for the seventh, and finally $\{\Box, \Box, \neg x_1, x_3\}$ for the eight. Therefore, the minimum number of unsatisfied clauses in ϕ is 2.

3.3.1 Soundness and completeness of clause MaxSAT tableaux

We prove that the minimum number of clauses that can be falsified in a multiset of clauses ϕ is m iff the cost of each completed clause MaxSAT tableau for ϕ is m.

Theorem 3.1. Soundness. Let ϕ be a multiset of clauses, and let T be a completed clause MaxSAT tableau for ϕ of cost m. Then, the minimum number of clauses that can be falsified in ϕ is m.

Proof. The clause MaxSAT tableau T was obtained by creating a sequence of clause MaxSAT tableaux $T_0, \ldots, T_n$ ($n \geq 0$) such that T_0 is an initial tableau for ϕ, $T_n = T$, and T_i was obtained by a single application of the extension or the contradiction rule on a branch of T_{i-1} for $i = 1, \ldots, n$. Assume that I is an optimal assignment of ϕ that falsifies k clauses, where $k \neq m$. By induction on n, we prove that the minimum number of active clauses that I falsifies among the branches of $T_0, \ldots, T_n$ (and in particular of T) is k:

Basis: T_0 has a single branch whose nodes are labelled with the clauses of ϕ, and such clauses are declared to be active in that branch. So, I falsifies k active clauses in T_0, and k is the minimum number of active clauses that can be falsified in T_0.

Inductive step: Assume that the minimum number of active clauses that I falsifies among the branches of T_{i-1} is k. We prove that the minimum number of active clauses that I falsifies among the branches of T_i is also k.

Since T_i was constructed from T_{i-1} by applying either the contradiction rule or the extension rule on a branch B of T_{i-1} and the rest of branches of T_{i-1} remain unchanged in T_i, we just need to prove that I satisfies the same number of active clauses in B and in at least one of the newly created branches, and does not decrease that number in the rest of newly created branches. We distinguish two cases:

- The contradiction rule was applied on B: two complementary unit clauses in B become inactive and an empty clause is added to the new branch B'. Since exactly one of the newly inactive unit clauses was falsified by I and we added one empty clause, I falsifies the same number of active clauses in B and B'.
- The extension rule was applied on B: If I satisfies the extended clause C of B, then I satisfies the leaf node of at least one of the newly created branches, say B'. The number of unsatisfied active clauses is preserved in B' and does not decrease in the rest of branches. If I falsifies C, then I falsifies the leaf nodes of all the newly created branches, and the number of unsatisfied active clauses is preserved in all these branches because C becomes inactive after the extension.

We proved that the minimum number of active clauses that I falsifies among the branches of $T_0, \ldots, T_n$ —and in particular of T— is k but this is in contradiction with T being a completed MaxSAT tableau for ϕ that has cost m: Since T is completed, the active clauses of any branch B of T with minimum cost is the union of a multiset with m empty clauses and a multiset of unit clauses whose complementary unit clauses do not occur in it. The multiset of unit clauses is clearly satisfiable, and so the minimum number of active clauses that can be falsified in B is m (not k), and is at least m in the rest of branches of T. Hence, the minimum number of clauses that can be falsified in ϕ is m. □

Theorem 3.2. Completeness. Let ϕ be a multiset of clauses whose minimum number of clauses that can be falsified is m. Then, each completed clause MaxSAT tableau for ϕ has cost m.

Proof. Each clause MaxSAT tableau for ϕ can be completed after a finite number of steps. This follows from the fact that the number of applications of extension rules in a branch is bound by the number of clauses in the input multiset, and the number of applications of the contradiction rules is bounded by the number of literals occurring in the input multiset.

Assume that there is a completed MaxSAT tableau T for ϕ that does not have cost m. We distinguish two cases:

(i) T has a branch B that has cost k, where $k < m$. Then, the active clauses of B are the union of a multiset with k empty clauses and a satisfiable multiset of unit clauses (otherwise, B could not be saturated because the contradiction rule could be applied). We define an assignment I of ϕ as follows: $I(x) = 1$ ($I(x) = 0$) if x ($\neg x$) is an active clause of B, and $I(x') = 0$ if variable x' does not occur in any active clause of B. We next prove that I satisfies at least $|\phi| - k$ clauses of ϕ, or equivalently I falsifies at most k clauses of ϕ. I is clearly an optimal assignment of B. If we undo all the applications of the contradiction rule in B, we get a branch B' whose active clauses form a multiset of unit clauses ϕ' that contains as many unit clauses as clauses are in ϕ, and each literal of each unit clause of ϕ' was derived from a different clause of ϕ. Since the clause of ϕ' are unit and there are k complementary pairs of unit clauses, I satisfies $|\phi| - k$ clauses of ϕ', and at least $|\phi| - k$ clauses of ϕ. We have therefore an assignment of ϕ that cannot falsify more than k clauses, but this is in contradiction with m being the minimum number of clauses that can be falsified in ϕ because $k < m$.

(ii) T has no branch of cost m. This is in contradiction with m being the minimum number of clauses that can be falsified in ϕ. Since the tableau rules preserve the minimum number of unsatisfied clauses, T must have a saturated branch of cost m.

Hence, each completed clause MaxSAT tableau T for a multiset of clauses ϕ has cost m if the minimum number of clauses that can be falsified in ϕ is m. $\square$

From the proof of Theorem 3.2, it follows that we can derive an optimal MaxSAT assignment I from a saturated branch B of minimum cost. The optimal assignment I sets a variable x to 1 (0) if B has a node labelled with the active clause x ($\neg x$); the rest of variables can be set to either 0 or 1.

3.3.2 Clause tableaux for WMaxSAT and WPMaxSAT

Many practical optimization problems admit more compact and natural MaxSAT encodings if they are encoded using weighted clauses instead of unweighted ones, as well as considering hard and soft clauses. To keep the description as simple as possible, we presented clause tableaux for Unweighted MaxSAT, but the proposed calculus can be extended to solve both WMaxSAT and WPMaxSAT.

In the case of WMaxSAT, we should keep in mind that a weighted clause (c, w) is equivalent to having w copies of the unweighted clause c. So, the application of the contradiction rule to two unit clauses $(l, w_1), (\neg l, w_2)$ amounts to adding an active empty clause with weight $w = min(w_1, w_2)$ (i.e.; $(\square, w)$), declare the clauses $(l, w_1), (\neg l, w_2)$ to be inactive, and add the active clauses $(l, w_1 - w)$ and $(\neg l, w_2 - w)$ in the newly created branch. Clauses with weight 0 are not added. The application of the extension rule to a weighted clause $(l_1 \vee \cdots \vee l_r, w)$, amounts to appending r nodes below the current branch, labelling each node with a different unit weighted clause from $\{(l_1, w), \ldots, (l_r, w)\}$. Finally, to get a complete calculus, we need to define a contraction rule: if a branch contains two active clauses (C, w_1) and (C, w_2), inactivate these clauses and add the active clause $(C, w_1 + w_2)$ in the newly created branch.

Example 3.3. Let $\phi = \{(\neg x_1, 3), (\neg x_2, 2), (x_1 \vee x_2, 2)\}$ be a multiset of weighted clauses. Figure 3.3 shows a completed clause WMaxSAT tableau T for ϕ. We first apply the extension rule to $(x_1 \vee x_2, 2)$ and derive two new branches. In the leftmost branch, the application of the contradiction rule to $(\neg x_1, 3)$ and $(x_1, 2)$ yields $(\square, 2)$ and $(\neg x_1, 1)$. In the rightmost branch, the application of the contradiction rule to $(\neg x_2, 2)$ and $(x_2, 2)$ yields $(\square, 2)$. The two saturated branches of the tableau have cost 2. The active clauses are: $\{(\neg x_2, 2), (\square, 2), (\neg x_1, 1)\}$ for the left branch and $\{(\neg x_1, 3), (\square, 2)\}$ for the right branch. Therefore, the minimum sum of weights of unsatisfied clauses in ϕ is 2.

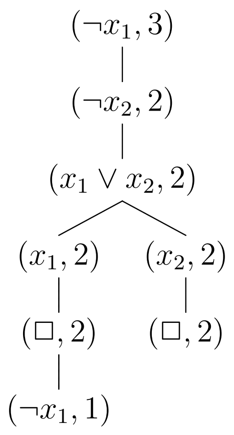


Figure 3.3: A completed clause WMaxSAT tableau for the multiset of weighted clauses $\phi = \{(\neg x_1, 3), (\neg x_2, 2), (x_1 \vee x_2, 2)\}$ that proves that the minimum sum of weights of unsatisfied clauses in ϕ is 2.

In the case of WPMaxSAT, we should add, to each hard clause, a weight greater than the sum of weights of the input soft clauses, and proceed as in the WMaxSAT case. Moreover, we should prune those branches in which a contradiction is detected between hard clauses or clauses derived from hard clauses, because they correspond to unfeasible solutions.

3.4 Clause tableaux for MinSAT

We showed, in the previous section, that the minimum number of empty clauses among the branches of a completed clause MaxSAT tableau for a multiset of clauses ϕ is equal to the number of clauses falsified by an optimal MaxSAT assignment of ϕ. Nevertheless, the maximum number of empty clauses among the branches of a completed clause MaxSAT tableau for ϕ is not the maximum number of clauses that can be falsified in ϕ; i.e., clause MaxSAT tableaux cannot solve MinSAT. This is so because the extension rule is unsound for MinSAT in the sense that it does not preserve the maximum number of clauses that can be falsified.

In the rest of the section, we first define a clause MinSAT tableau calculus that incorporates a sound extension rule. We then prove the soundness and completeness of the proposed calculus. Finally, we present how our results can be extended to deal with WMinSAT and WPMinSAT instances. Note that, in MinSAT, we also need to derive contradictions from satisfiable instances because the maximum number of clauses that can be falsified in a satisfiable instance other than the empty multiset is always greater than or equal to one.

Definition 3.4. A clause MinSAT tableau is a finite tree whose nodes are labelled with multisets of clauses. A branch is a maximal path in a tree.

Note that we now label the nodes of a tableau with multisets of clauses instead of clauses. Also note that clause MinSAT tableaux do not need to declare clauses either as active or inactive.

Given a multiset of clauses ϕ, where $l_1, \ldots, l_r$ are the literals occurring in ϕ, we recall that the instantiation of $l_1, \ldots, l_r$ in ϕ, denoted by $\phi_{l_1|\cdots|l_r}$, is the multiset of clauses resulting of eliminating all the occurrences of $\neg l_1, \ldots, \neg l_r$ from ϕ and removing all the clauses with occurrences of $l_1, \ldots, l_r$.

Definition 3.5. Let ϕ be a multiset of clauses. A clause MinSAT tableau for ϕ is constructed by a sequence of applications of the following rules:

Initialize A tree with a single branch with a single node labelled with the multiset of clauses ϕ is a clause MinSAT tableau for ϕ. Such a tableau is called initial tableau.

Extension Given a clause MinSAT tableau T for ϕ, and a branch B of T whose leaf node is labelled with a multiset $\phi' = \phi'' \cup \{l_1 \vee \cdots \vee l_r\}$, the tableau obtained by appending a new left node below B labelled with the multiset $\phi'_{\neg l_1|\cdots|\neg l_r}$ and a new right node below B labelled with the multiset ϕ'' is a clause MinSAT tableau for ϕ.

In the definition of the extension rule, note that $\phi'_{\neg l_1|\cdots|\neg l_r} = \{\square\} \cup \phi''_{\neg l_1|\cdots|\neg l_r}$.

Definition 3.6. Let T be a clause MinSAT tableau for a multiset of clauses ϕ, and let B be a branch of T. Branch B is saturated iff its leaf node is labelled with the empty multiset or with a multiset of empty clauses. Tableau T is completed iff all its branches are saturated. The cost of a saturated branch is the number of empty clauses in its leaf node. The cost of a completed clause MinSAT tableau is the maximum cost among all its branches.

Example 3.4. Let $\phi_1 = \{\neg x_1, \neg x_2, x_1 \vee x_2\}$ and $\phi_2 = \{x_1 \vee x_2, \neg x_1 \vee x_3, \neg x_2 \vee \neg x_3\}$ be multisets of clauses. Figures 3.4 and 3.5 show completed clause MinSAT tableaux for ϕ_1 and ϕ_2, respectively. The leaf nodes of the branches of the tableau for ϕ_1 have at most cost 2, and of the tableau for ϕ_2 have at most cost 1. Therefore, the maximum number of clauses that can be falsified is 2 in ϕ_1 and 1 in ϕ_2. Note that ϕ_2 is satisfiable.

3.4.1 Soundness and completeness of clause MinSAT tableaux

We first prove that the extension rule preserves the maximum number of unsatisfied clauses among the branches of a clause MinSAT tableau and then the soundness and completeness of the clause MinSAT tableau calculus.

Lemma 3.1. Let $\phi = \phi' \cup \{l_1 \vee \cdots \vee l_r\}$ be a multiset of clauses, and let $minsat(\psi)$ denote the maximum number of clauses that can be falsified in the multiset of clauses ψ. It holds that $minsat(\phi) = \max(minsat(\phi'), minsat(\phi_{\neg l_1|\cdots|\neg l_r}))$.

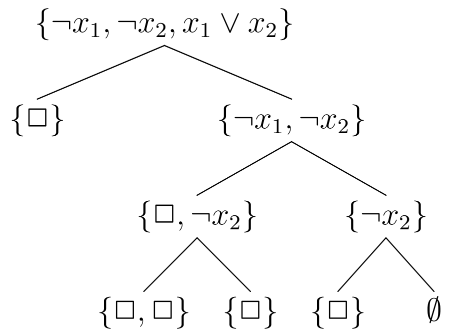


Figure 3.4: Completed clause MinSAT tableau for $\phi_1 = \{\neg x_1, \neg x_2, x_1 \vee x_2\}$.

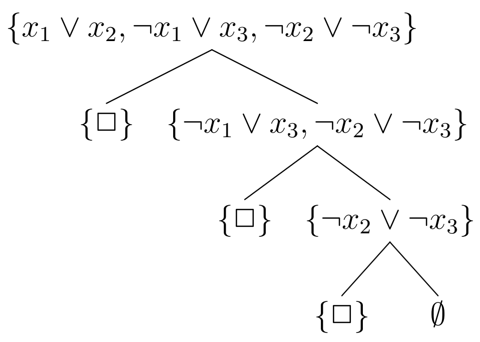

$\{x_1 \vee x_2, \neg x_1 \vee x_3, \neg x_2 \vee \neg x_3\}$
$\{\square\}$
$\{\neg x_1 \vee x_3, \neg x_2 \vee \neg x_3\}$
$\{\square\}$
$\{\neg x_2 \vee \neg x_3\}$
$\{\square\}$
$\emptyset$

Figure 3.5: Completed clause MinSAT tableau for $\phi_2 = \{x_1 \vee x_2, \neg x_1 \vee x_3, \neg x_2 \vee \neg x_3\}$.

Proof. We first prove the following pair of inequalities: $minsat(\phi) \geq minsat(\phi')$ and $minsat(\phi) \geq minsat(\phi_{\neg l_1|\cdots|\neg l_r})$.

Since $\phi' \subset \phi$ and the maximum number of clauses that can be falsified in every subset of ϕ cannot be greater than the maximum number of clauses that can be falsified in ϕ, it holds that $minsat(\phi) \geq minsat(\phi')$.

Assume that there is an optimal assignment I' of $\phi_{\neg l_1|\cdots|\neg l_r}$ that falsifies more clauses than an optimal assignment I of ϕ. Then, we could extend I' by setting $I'(l_i) = 0$ for $i = 1, \ldots, r$, and get an optimal assignment of ϕ that falsifies more clauses than I: If we restore the occurrences of the literals $l_1, \ldots, l_r$ in the clauses of $\phi_{\neg l_1|\cdots|\neg l_r}$ in which such literals were eliminated when $\neg l_1, \ldots, \neg l_r$ were instantiated in ϕ, we get a multiset ϕ'' such that $\phi'' \subseteq \phi$. It holds that the number of clauses that I' falsifies in $\phi_{\neg l_1|\cdots|\neg l_r}$ and ϕ'' is the same because I' falsifies the added literals, but this is in contradiction with I being optimal. Therefore, $minsat(\phi) \geq minsat(\phi_{\neg l_1|\cdots|\neg l_r})$.

Taking into account the previous inequalities, we proceed to prove now the equality $minsat(\phi) = \max(minsat(\phi'), minsat(\phi_{\neg l_1|\cdots|\neg l_r}))$. Let I be an optimal assignment of ϕ. We distinguish two cases:
i) I satisfies $l_1 \vee \cdots \vee l_r$. Then, I falsifies the same clauses in ϕ and ϕ', and is also an optimal assignment of ϕ' because $minsat(\phi) \geq minsat(\phi')$. Because it holds that $minsat(\phi) = minsat(\phi')$, and also $minsat(\phi) \geq minsat(\phi_{\neg l_1|\cdots|\neg l_r})$, it follows that $minsat(\phi) = \max(minsat(\phi'), minsat(\phi_{\neg l_1|\cdots|\neg l_r}))$.
ii) I falsifies $l_1 \vee \cdots \vee l_r$. Then, I sets $l_1, \ldots, l_r$ to 0 and I falsifies the same number of clauses in ϕ and $\phi_{\neg l_1|\cdots|\neg l_r}$. Since $minsat(\phi) \geq minsat(\phi_{\neg l_1|\cdots|\neg l_r})$, it follows that I is also an optimal assignment of $\phi_{\neg l_1|\cdots|\neg l_r}$. Since $minsat(\phi) = minsat(\phi_{\neg l_1|\cdots|\neg l_r})$ and $minsat(\phi) \geq minsat(\phi')$, then $minsat(\phi) = \max(minsat(\phi'), minsat(\phi_{\neg l_1|\cdots|\neg l_r}))$. □

Theorem 3.3. Soundness. Let ϕ be a multiset of clauses, and let T be a completed clause MinSAT tableau for ϕ that has cost m. Then, the maximum number of clauses that can be falsified in ϕ is m.

Proof. The clause MinSAT tableau T was obtained by creating a sequence of clause MinSAT tableaux $T_0, \ldots, T_n$ ($n \geq 0$) such that T_0 is an initial tableau for ϕ, $T_n = T$, and T_i was obtained by a single application of the extension rule on a leaf node of a branch of T_{i-1} for $i = 1, \ldots, n$. Assume that I is an optimal assignment of ϕ that falsifies k clauses, where $k \neq m$. By induction on n, we prove that the maximum number of clauses that I falsifies among the leaf nodes of the branches of $T_0, \ldots, T_n$ (and in particular of T) is k:

Basis: T_0 has a single branch with one node labelled with the clauses of ϕ. So, I falsifies k clauses in T_0, and k is the maximum number of clauses that can be falsified in T_0.

Inductive step: Assume that the maximum number of clauses that I falsifies among the leaf nodes of the branches of T_{i-1} is k. We prove that the maximum number of clauses that I falsifies among the leaf nodes of the branches of T_i is also k.

T_i was constructed from T_{i-1} by applying the extension rule on a branch B of T_{i-1}. If I falsifies k clauses of the leaf node of B, by Lemma 3.1, the maximum number of clauses that I falsifies among the branches of T_i remains k. If I falsifies r clauses of the leaf node of B, where $r < k$, then I falsifies k clauses of the leaf node of a branch B' of T_i ($B' \neq B$ and B' is also a branch of T_{i-1}), and I cannot falsify more than k clauses in the leaf nodes of any of the two branches derived from B because otherwise we could define an assignment that falsifies more than k clauses of the leaf node of B.

We proved that the maximum number of clauses that I falsifies among the leaf nodes of the branches of $T_0, \ldots, T_n$ —and in particular of T— is k, but this is in contradiction with T being a completed clause MinSAT tableau for ϕ that has cost m. Since T is completed and has cost m, the leaf nodes are labelled with either a multiset of empty clauses or the empty formula, and there is at least a branch B whose leaf node is a multiset with m empty clauses. So, the maximum number of clauses that can be falsified in the leaf node of B is m (and not k), and is at most m in the rest of leaf nodes of branches of T. Hence, the maximum number of clauses that can be falsified in ϕ is m. □

Theorem 3.4. Completeness. Let ϕ be a multiset of clauses whose maximum number of clauses that can be falsified in ϕ is m. Then, any completed clause MinSAT tableau for ϕ has cost m.

Proof. Each clause MinSAT tableau T for ϕ can be completed after a finite number of steps. This follows from the fact that the extension rule either eliminates one clause or replaces one clause with an empty clause at each application of the rule. Moreover, the instantiation of literals neither increases the number of clauses nor increases the number of literals per clause. Thus, after a finite number of applications of the extension rule, T is transformed into a completed clause MinSAT tableau.

Assume that there is a completed clause MinSAT tableau T for ϕ that does not have cost m. We distinguish two cases:

(i) T has a branch B that has cost k, where $k > m$. Then, the leaf node of B has k empty clauses, and each empty clause is derived from a clause of ϕ; let $C_1, \ldots, C_k$ be such clauses. We define an assignment I of ϕ as follows: $I(x) = 1$ ($I(x) = 0$) if $\neg x$ (x) occurs in $\{C_1, \ldots, C_k\}$; and $I(x) = 0$ if variable x does not occur in $\{C_1, \ldots, C_k\}$. Note that $\{C_1, \ldots, C_k\}$ only contain literals with both the same variable and polarity because the corresponding literals with opposite polarity occur in clauses that were eliminated. Assignment I falsifies at least k clauses of ϕ because each literal occurring in $\{C_1, \ldots, C_k\}$ is unsatisfied by I. Since $k > m$, this is in contradiction with m being the maximum number of clauses that can be falsified in ϕ.

(ii) T has no branch of cost m. This is in contradiction with m being the maximum number of clauses that can be falsified in ϕ. Since an optimal assignment falsifies m clauses of the initial tableau and the leaf nodes of a completed clause MinSAT tableau are labelled with either a multiset of empty clauses or the empty formula, by Lemma 3.1, T must have a branch of cost m. □

From the proof of Theorem 3.4 it follows that, for building an optimal assignment I from a completed tableau, we have to consider a branch with a maximum number of empty clauses in its leaf node and identify the input clauses that became empty. Then, for each one of such clauses, say $l_1 \vee \cdots \vee l_m$, we define $I(l_i) = 0$ for $i = 1, \ldots, m$; and the variables that do not appear in such clauses can be set to an arbitrary value. For example, in the tableau for $\phi_1 = \{\neg x_1, \neg x_2, x_1 \vee x_2\}$ of Figure 3.4, the input clauses that became empty in the branch with maximum cost are $\{\neg x_1, \neg x_2\}$ and, therefore, $I(x_1) = I(x_2) = 1$ is an optimal assignment of ϕ_1.

3.4.2 Clause tableaux for WMinSAT and WPMinSAT

We presented clause tableaux for Unweighted MinSAT to keep the description as simple as possible. We now describe how the clause MinSAT tableau calculus can be extended to deal with WMinSAT and WPMinSAT instances.

In the case of WMinSAT, we use the same tableau rules but keeping the weights of the clauses. Note that the extension rule either removes clauses or eliminates literals. When a literal is eliminated from a clause, the shortened clause maintains the same weight. In addition, we also could need to collapse several weighted clauses of the form $(C, w_1), \ldots, (C, w_k)$ into a single weighted clause $(C, w_1 + \cdots + w_k)$.

Example 3.5. Let $\phi = \{(x_1, 1), (\neg x_2, 3), (x_1 \vee \neg x_2, 5), (x_1 \vee \neg x_3, 2), (x_2 \vee x_3, 1)\}$ be a multiset of weighted clauses. Figure 3.6 shows a completed clause WMinSAT tableau T for ϕ. The leaf nodes of the branches of T have at most cost 11. Therefore, the maximum sum of the weights of the clauses that can be falsified in ϕ is 11.

In the case of WPMinSAT, we must first derive an equivalent WMinSAT instance and then solve the derived instance as explained above. We will assume that there is an assignment that satisfies all the hard clauses, since otherwise no feasible solution exists.

Given a WPMinSAT instance ϕ whose number of hard clauses is $\#hard$ and whose sum of the weights of all its soft clauses is w, we derive a WMinSAT instance ϕ' by adding (i) all the soft clauses in ϕ, and (ii) the soft clauses $(\neg l_1, w+1)$, $(l_1 \vee \neg l_2, w+1)$, $\ldots$, $(l_1 \vee l_2 \vee \cdots \vee \neg l_k, w+1)$ for each hard clause $h = l_1 \vee l_2 \vee \cdots \vee l_k$ in ϕ.

Observe that an assignment I satisfies h iff I falsifies exactly one clause among $\neg l_1, l_1 \vee \neg l_2, \ldots, l_1 \vee l_2 \vee \cdots \vee \neg l_k$; or equivalently, I falsifies h iff I satisfies all these clauses. Since the clauses derived from hard clauses have weight $w+1$ and we assumed that the hard part of ϕ is satisfiable, every optimal solution of ϕ' falsifies exactly one clause derived from a hard clause and is also an optimal solution of ϕ. Besides, if the maximum sum of the weights of the unsatisfied clauses in ϕ' is m, then the maximum sum of the weights of the unsatisfied clauses in ϕ is $m - \#hard \times (w+1)$. The treatment of hard clauses in MinSAT tableaux is not as in MaxSAT tableaux, where it is enough to add the weight $w+1$ to each hard clause and solve the resulting WMaxSAT instance.

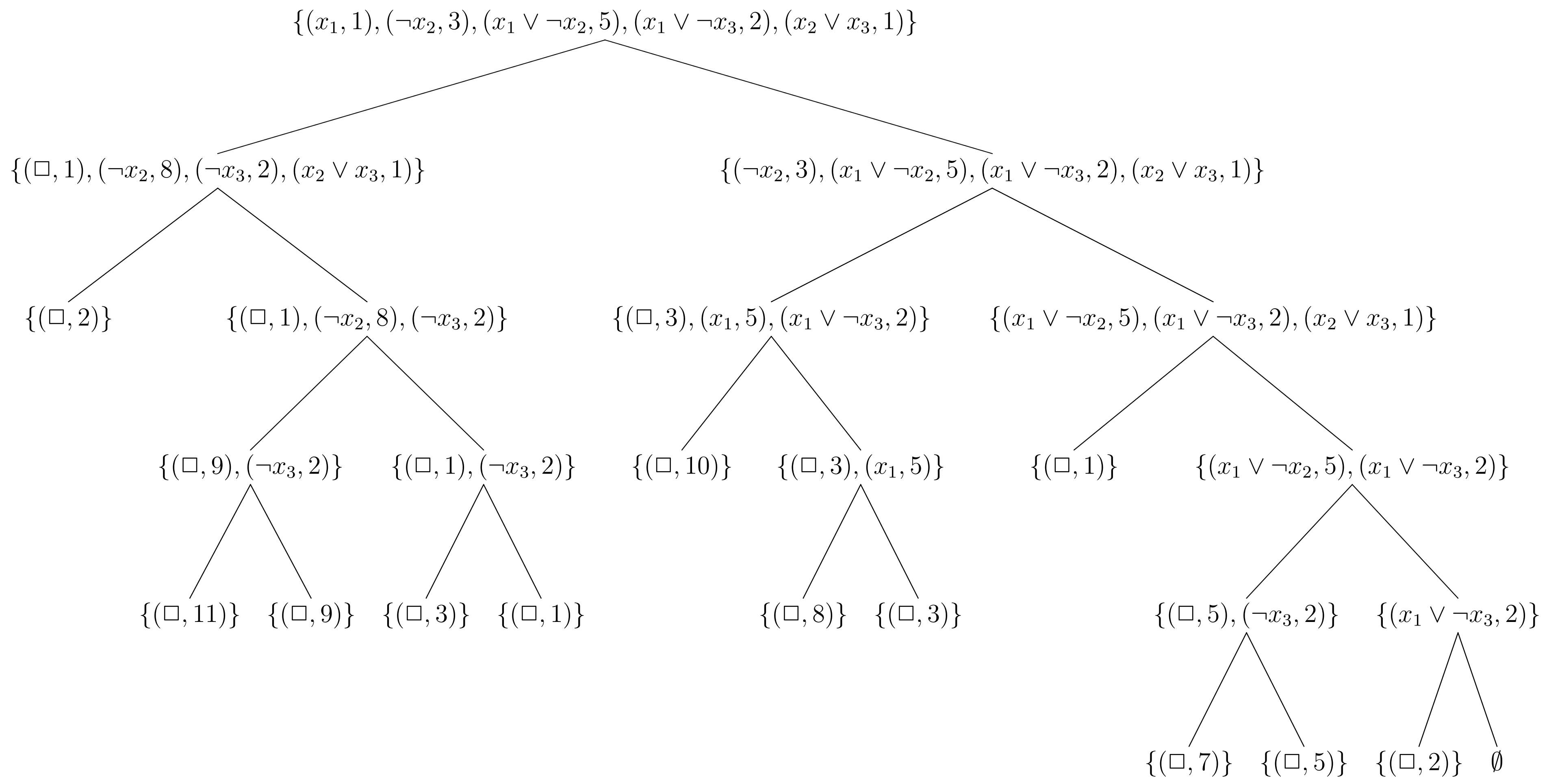

$\{(x_1, 1), (\neg x_2, 3), (x_1 \vee \neg x_2, 5), (x_1 \vee \neg x_3, 2), (x_2 \vee x_3, 1)\}$
$\{(\Box, 1), (\neg x_2, 8), (\neg x_3, 2), (x_2 \vee x_3, 1)\}$
$\{(\neg x_2, 3), (x_1 \vee \neg x_2, 5), (x_1 \vee \neg x_3, 2), (x_2 \vee x_3, 1)\}$
$\{(\Box, 2)\}$
$\{(\Box, 1), (\neg x_2, 8), (\neg x_3, 2)\}$
$\{(\Box, 3), (x_1, 5), (x_1 \vee \neg x_3, 2)\}$
$\{(x_1 \vee \neg x_2, 5), (x_1 \vee \neg x_3, 2), (x_2 \vee x_3, 1)\}$
$\{(\Box, 9), (\neg x_3, 2)\}$
$\{(\Box, 1), (\neg x_3, 2)\}$
$\{(\Box, 10)\}$
$\{(\Box, 3), (x_1, 5)\}$
$\{(\Box, 1)\}$
$\{(x_1 \vee \neg x_2, 5), (x_1 \vee \neg x_3, 2)\}$
$\{(\Box, 11)\}$
$\{(\Box, 9)\}$
$\{(\Box, 3)\}$
$\{(\Box, 1)\}$
$\{(\Box, 8)\}$
$\{(\Box, 3)\}$
$\{(\Box, 5), (\neg x_3, 2)\}$
$\{(x_1 \vee \neg x_3, 2)\}$
$\{(\Box, 7)\}$
$\{(\Box, 5)\}$
$\{(\Box, 2)\}$
$\emptyset$

Figure 3.6: Completed clause WMinSAT tableau for $\phi = \{(x_1, 1), (\neg x_2, 3), (x_1 \vee \neg x_2, 5), (x_1 \vee \neg x_3; 2), (x_2 \vee x_3, 1)\}$.

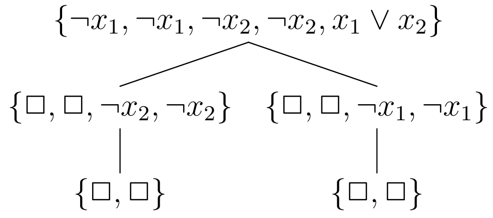

$\{\neg x_1, \neg x_1, \neg x_2, \neg x_2, x_1 \vee x_2\}$
$\{\Box, \Box, \neg x_2, \neg x_2\}$
$\{\Box, \Box, \neg x_1, \neg x_1\}$
$\{\Box, \Box\}$
$\{\Box, \Box\}$

Figure 3.7: Proof tree for $\{\neg x_1, \neg x_1, \neg x_2, \neg x_2, x_1 \vee x_2\}$ using SAT clause branching.

3.5 Clause tableaux for MaxSAT and MinSAT

After defining a clause tableau calculus for MaxSAT and another for MinSAT, a natural question is whether there is a calculus that is valid for both MaxSAT and MinSAT.

In this section, we propose a tableau calculus for MaxSAT and MinSAT that preserves the number of unsatisfied clauses in the generated subproblems adequately. The leaf nodes of a completed tableau contain a number of empty clauses ranging between the minimum and the maximum number of unsatisfied clauses in the input formula, and there is at least one branch with the minimum value and at least one branch with the maximum value. This scheme also generates optimal MaxSAT and MinSAT assignments by inspecting the optimal branches.

A natural extension of clause tableaux is to perform some kind of local inference at each node, as MinSAT tableaux do in one of the branches when assigning the variables occurring in a select clause to force its violation. In SAT, assuming that the initial tableau contains a single node with the input set of clauses, we can introduce local inference by applying the following extension rule: If the leaf node of a branch B is labelled with the set of clauses $\phi = \phi' \cup \{l_1 \vee \cdots \vee l_k\}$, then append k sibling nodes below B labelled with $\phi_{l_1}, \ldots, \phi_{l_k}$. Then, the input set of clauses is unsatisfiable iff the empty clause has been derived in each branch. Actually, if the local inference applied is unit propagation instead of the instantiation of literal l_i (i.e., ϕ_{l_i}), also known as unit clause rule, we get the DPLL procedure [80] with clause branching [101].

Unfortunately, the outlined SAT approach is unsound for MaxSAT and MinSAT because it does not preserve neither the maximum nor the minimum number of unsatisfied clause. For example, if we consider the multiset of clauses $\phi = \{\neg x_1, \neg x_1, \neg x_2, \neg x_2, x_1 \vee x_2\}$, the generated tableau has two leaf nodes with two empty clauses, as Figure 3.7 shows. Note that we first branch on x_1 (labelling the node with ϕ_{x_1}) and x_2 (labelling the node with ϕ_{x_2}) and then instantiate $\neg x_2$ in the left branch and $\neg x_1$ in the right branch. However, the MaxSAT solution of ϕ is one and the MinSAT solution is four. Also note that branching on $l_1, \ldots, l_k$ is sound in MaxSAT tableaux, but becomes unsound when local inference is added to each node.

The key point to integrate MaxSAT and MinSAT is to define an extension rule that preserves both the maximum and the minimum number of unsatisfied clauses. This is what we do in the clause MaxMinSAT tableau defined below.

Definition 3.7. A clause MaxMinSAT tableau is a finite tree whose nodes are labelled with multisets of clauses. A branch is a maximal path in a tree.

Given a multiset of clauses ϕ, where $l_1, \ldots, l_r$ are the literals occurring in ϕ, we recall again, in this section, that the instantiation of $l_1, \ldots, l_r$ in ϕ, denoted by $\phi_{l_1|\cdots|l_r}$, is the multiset of clauses resulting of eliminating all the occurrences of $\neg l_1, \ldots, \neg l_r$ from ϕ and removing all the clauses with occurrences of $l_1, \ldots, l_r$.

Definition 3.8. Let ϕ be a multiset of clauses. A clause MaxMinSAT tableau for ϕ is constructed by a sequence of applications of the following rules:

Initialize A tree with a single branch with a single node labelled with the multiset of clauses ϕ is a clause MaxMinSAT tableau for ϕ. Such a tableau is called initial tableau.

Extension Given a clause tableau T for ϕ, and a branch B of T whose leaf node is labelled with a multiset $\phi = \phi' \cup \{l_1 \vee \cdots \vee l_k\}$, the tableau obtained by appending $k+1$ new left nodes below B labelled with the multisets $\phi_{l_1}, \ldots, \phi_{l_k}, \phi_{\neg l_1, \ldots, \neg l_k}$ is a clause MaxMinSAT tableau for ϕ.

Definition 3.9. Let T be a clause MaxMinSAT tableau for a multiset of clauses ϕ, and let B be a branch of T. Branch B is saturated iff its leaf node is labelled with the empty multiset or with a multiset of empty clauses. Tableau T is completed iff all its branches are saturated. The cost of a saturated branch is the number of empty clauses in its leaf node. The MaxSAT cost of a completed clause MaxMinSAT tableau is the minimum cost among all its branches, and the MinSAT cost of a completed clause MaxMinSAT tableau is the maximum cost among all its branches.

We prove below that the branches with MaxSAT and MinSAT costs provide optimal MaxSAT and MinSAT solutions, respectively.

Example 3.6. We consider again the multiset of clauses $\phi = \{\neg x_1, \neg x_1, \neg x_2, \neg x_2, x_1 \vee x_2\}$ of Figure 3.7. Figure 3.8 displays a completed clause MaxMinSAT tableau for ϕ. The MaxSAT cost of the tableau is 1 and the MinSAT cost is 4. Hence, the minimum number of clauses that can be falsified in ϕ is 1 and the maximum number is 4.

The next two lemmas prove that the MaxMinSAT extension rule preserves both the maximum and the minimum number of unsatisfied clauses. In other words, we prove its soundness. Based on these results, we then prove the completeness of the clause MaxMinSAT tableau calculus.

Lemma 3.2. MaxSAT clause branching. Let ϕ be a multiset of clauses, let $l_1 \vee \cdots \vee l_k$ be a clause of ϕ, and let $\texttt{maxsat}(\phi)$ be the minimum number of unsatisfied clauses in ϕ. Then,

$$\texttt{maxsat}(\phi) = \min(\texttt{maxsat}(\phi_{l_1}), \ldots, \texttt{maxsat}(\phi_{l_k}), \texttt{maxsat}(\phi_{\neg l_1, \ldots, \neg l_k})) \quad (3.1)$$

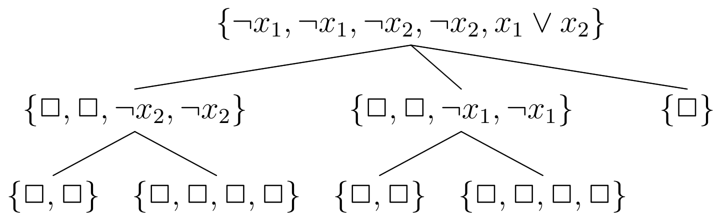

$\{\neg x_1, \neg x_1, \neg x_2, \neg x_2, x_1 \vee x_2\}$
$\{\Box, \Box, \neg x_2, \neg x_2\}$
$\{\Box, \Box, \neg x_1, \neg x_1\}$
$\{\Box\}$
$\{\Box, \Box\}$
$\{\Box, \Box, \Box, \Box\}$
$\{\Box, \Box\}$
$\{\Box, \Box, \Box, \Box\}$

Figure 3.8: A completed clause MaxMinSAT tableau for $\{\neg x_1, \neg x_1, \neg x_2, \neg x_2, x_1 \vee x_2\}$.

Proof. Let I be an optimal MaxSAT assignment. We prove that the minimum number of unsatisfied clauses in ϕ is the same as the minimum number of unsatisfied clauses in at least one of the multisets $\phi_{l_1}, \ldots, \phi_{l_k}, \phi_{\neg l_1, \ldots, \neg l_k}$, and is not smaller in the rest of multisets. We distinguish two cases:

1. I satisfies $l_1 \vee \cdots \vee l_k$: If I satisfies l_i, $1 \leq i \leq k$, then $\texttt{maxsat}(\phi) = \texttt{maxsat}(\phi_{l_i})$ because deleting the clauses containing l_i and removing the occurrences of $\neg l_i$ preserves the number of unsatisfied clauses between ϕ and ϕ_{l_i} for every assignment. Since at least one literal l_i is satisfied by I, the minimum number of unsatisfied clauses is preserved in one of the derived multisets.

   If I does not satisfy l_j, $1 \leq j \leq k$, then $\texttt{maxsat}(\phi) \leq \texttt{maxsat}(\phi_{l_j})$. Assume that there exists an assignment I' of ϕ_{l_j} that falsifies less than $\texttt{maxsat}(\phi)$ clauses. If we extend I' by assigning $I'(l_j) = true$, we get an assignment of ϕ that falsifies less than $\texttt{maxsat}(\phi)$ clauses. But this is in contradiction with I being optimal. So, it holds that $\texttt{maxsat}(\phi) \leq \texttt{maxsat}(\phi_{l_j})$.

   Finally, we have to prove that $\texttt{maxsat}(\phi) \leq \texttt{maxsat}(\phi_{\neg l_1, \ldots, \neg l_k})$. Observe that $\texttt{maxsat}(\phi_{\neg l_1, \ldots, \neg l_k}) = \texttt{maxsat}(\ldots(\texttt{maxsat}(\texttt{maxsat}(\phi_{\neg l_1})_{\neg l_2})\ldots)_{\neg l_k})$. The literals $\neg l_i$, $1 \leq i \leq k$, that are satisfied by I preserve the number of unsatisfied clauses, and the literals $\neg l_j$, $1 \leq j \leq k$, that are not satisfied by I can increase the number of unsatisfied clauses. Since I satisfies $l_1 \vee \cdots \vee l_k$, I does not satisfy at least one literal in $\{\neg l_1, \ldots, \neg l_k\}$ and therefore $\texttt{maxsat}(\phi) \leq \texttt{maxsat}(\phi_{\neg l_1, \ldots, \neg l_k})$.

   The last two cases guarantee that the number of unsatisfied clauses does not decrease in any case.

2. I does not satisfy $l_1 \vee \cdots \vee l_k$: In this case, I does not satisfy any literal in the clause. As shown above, $\texttt{maxsat}(\phi) \leq \texttt{maxsat}(\phi_{l_i})$ for every i, $1 \leq i \leq k$. On the other hand, $\texttt{maxsat}(\phi) = \texttt{maxsat}(\phi_{\neg l_1, \ldots, \neg l_k})$ because I satisfies $\neg l_1, \ldots, \neg l_k$, and deleting the clauses containing $\neg l_j$ and removing the occurrences of l_j preserves the number of unsatisfied clauses in this case.

$\Box$

Lemma 3.3. MinSAT clause branching. Let ϕ be a multiset of clauses, let $l_1 \vee \cdots \vee l_k$ be a clause of ϕ, and let $\texttt{minsat}(\phi)$ be the maximum number of unsatisfied clauses in ϕ.

Then,

$$\texttt{minsat}(\phi) = \max(\texttt{minsat}(\phi_{l_1}), \dots, \texttt{minsat}(\phi_{l_k}), \texttt{minsat}(\phi_{\neg l_1,\dots,\neg l_k})) \quad (3.2)$$

Proof. Let I be an optimal MinSAT assignment. We can prove, using the arguments of Lemma 3.2, that the maximum number of unsatisfied clauses in ϕ is the same as the maximum number of unsatisfied clauses in at least one of the multisets $\phi_{l_1}, \dots, \phi_{l_k}, \phi_{\neg l_1,\dots,\neg l_k}$, and is not greater in the rest of multisets. We have to take into account the following facts: (i) If I satisfies a literal l_i, $1 \leq i \leq k$, then ϕ_{l_i} clearly preserves the number of unsatisfied clauses and so $\texttt{minsat}(\phi) = \texttt{minsat}(\phi_{l_i})$. (ii) If I does not satisfy l_j, $1 \leq j \leq k$, then $\texttt{minsat}(\phi) \geq \texttt{minsat}(\phi_{l_j})$: Assume that there exists an assignment I' of ϕ_{l_j} that falsifies more than $\texttt{minsat}(\phi)$ clauses. If we extend I' by assigning $I'(l_j) = true$, we get an assignment of ϕ that falsifies more than $\texttt{minsat}(\phi)$ clauses. But this is in contradiction with I being optimal. So, it holds that $\texttt{minsat}(\phi) \geq \texttt{minsat}(\phi_{l_j})$. □

Theorem 3.5. A completed clause MaxMinSAT tableau for a multiset of clauses ϕ provides an optimal MaxSAT assignment and an optimal MinSAT assignment.

Proof. The root node of a completed MaxMinSAT tableau is labelled with the input formula. Such a tableau is finite because each descendant has at least one variable less than its parents: At least one less variable in the descendants labelled with $\phi_{l_1}, \dots, \phi_{l_k}$, and at least k less variables in the descendants labelled with $\phi_{\neg l_1,\dots,\neg l_k}$. Thus, after a finite number of steps, the leaves of a tableau contain either the empty formula or a multiset of empty clauses. By Lemma 3.2, all the branches with the minimum number of empty clauses correspond to optimal MaxSAT solutions. By Lemma 3.3, all the branches with the maximum number of empty clauses correspond to optimal MinSAT solutions. Besides, Lemma 3.2 and Lemma 3.3 guarantee that the cost of each branch ranges between the maximum and minimum number of unsatisfied clauses in ϕ.

The clauses became empty because of the literals that were instantiated at each node, which allowed to remove the complementary literals. Thus, in the optimal branches, the assignments that set those literals to true and the rest of literals appearing in the input formula to an arbitrary value are optimal assignments. □

All the results of this section also hold if we replace the proposed extension rule with the following rule: *If the leaf node of a branch B is labelled with the multiset $\phi = \phi' \cup \{l_1 \vee \cdots \vee l_k\}$, then append $k+1$ sibling nodes below B labelled with ϕ_{l_1}, $\phi_{\neg l_1, l_2}, \dots, \phi_{\neg l_1,\dots,\neg l_{k-1}, l_k}$ and $\phi_{\neg l_1,\dots,\neg l_k}$.* This result can be proved with the same arguments of Lemmas 3.2 and 3.3.

The extension of our approach to Weighted MaxSAT/MinSAT and Weighted Partial MaxSAT/MinSAT is as in BnB MaxSAT and MinSAT algorithms [7, 138, 148]. Roughly speaking, in Weighted MaxSAT/MinSAT, we just need to propagate weights and, given a multiset of empty weighted clauses $\{(\Box, w_1), \dots, (\Box, w_k)\}$, we replace it with $\{(\Box, w_1 + \cdots + w_k)\}$. In Weighted Partial MaxSAT/MinSAT, we can apply the same inference as in SAT in the hard part.

3.6 Concluding remarks

We have defined three complete logical calculi for MaxSAT and MinSAT. The first one is a tableau calculus for MaxSAT, the second one is a tableau calculus for MinSAT, and the third one is a tableau calculus that is valid for both MaxSAT and MinSAT.

These results will be the starting point for defining a genuine calculus for non-clausal MaxSAT in the next chapter. The contributions of this chapter have inspired a natural deduction MaxSAT calculus [66].

Chapter 4

Solving Non-Clausal MaxSAT and Non-Clausal MinSAT

This chapter proposes two approaches to solving non-clausal MaxSAT and non-clausal MinSAT: In the first approach, it defines three different cost-preserving clausal form transformations –called direct, improved and Tseitin-based transformations– from non-clausal MaxSAT/MinSAT to clausal MaxSAT/MinSAT. These transformations reduce non-clausal MaxSAT/MinSAT to clausal MaxSAT/MinSAT so that clausal MaxSAT/MinSAT solvers can be used to solve the MaxSAT problem of multisets of propositional formulas that are not necessarily in clausal form. In the second approach, it defines a genuine tableau calculus for non-clausal MaxSAT, proves its soundness and completeness, and describes how it can be extended to deal with hard and soft formulas. Moreover, it describes how the tableau calculus for non-clausal MaxSAT can be used to solve non-clausal MinSAT.

The presentation closely follows the work published in two conference papers [142, 143]. The cost-preserving clausal form transformations appeared first in [142], and the tableau calculus for non-clausal MaxSAT appeared first in [143].

4.1 Introduction

We can distinguish between clausal MaxSAT/MinSAT and non-clausal MaxSAT/MinSAT. Clausal MaxSAT, known simply as MaxSAT, is to find an assignment that minimizes the number of unsatisfied clauses in a given multiset of clauses, and non-clausal MaxSAT is to find an assignment that minimizes the number of unsatisfied formulas in a given multiset of propositional formulas that are not necessarily in clausal form. Similarly, we distinguish between clausal MinSAT and non-clausal MinSAT.

MaxSAT and MinSAT solvers require the input to be a multiset of clauses and return an assignment that minimizes/maximizes the number of unsatisfied clauses in the input multiset. However, many problems in real-world applications are encoded as MaxSAT/MinSAT instances consisting of a multiset of propositional formulas that are not necessarily clauses. Such encodings cannot be solved with modern MaxSAT/Min-

SAT solvers.

For solving a MaxSAT/MinSAT instance consisting of a multiset of propositional formulas, we have two options:

- Transform the input propositional formula to clausal form using a cost-preserving transformation and then solve the resulting instance with a clausal MaxSAT/MinSAT solver.

- Define a genuine MaxSAT/MinSAT proof procedure that directly deals with arbitrary propositional formulas.

In this chapter, we first define three different cost-preserving clausal form transformations –called direct, improved and Tseitin-based transformations– from non-clausal MaxSAT/MinSAT to clausal MaxSAT/MinSAT. These transformations reduce non-clausal MaxSAT/MinSAT to clausal MaxSAT/MinSAT so that clausal MaxSAT/MinSAT solvers can be used to solve the MaxSAT/MinSAT problem of multisets of arbitrary propositional formulas. We then define a genuine tableau calculus for non-clausal MaxSAT, prove its soundness and completeness, and describe how it can be extended to deal with Weighted Partial MaxSAT instances. We also describe how the tableau calculus for non-clausal MaxSAT can be used to solve non-clausal MinSAT.

Before presenting the contributions, we give some background definitions, in the following three paragraphs, to make the chapter as self-contained as possible.

A propositional formula is an expression constructed from propositional variables using the propositional connectives $\wedge, \vee, \rightarrow$ and $\neg$ in accordance with the following rules: i) each propositional variable is a propositional formula; and ii) if A and B are propositional formulas, then so are $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, and $(\neg A)$. A non-clausal MaxSAT/MinSAT instance is a multiset of propositional formulas.

A truth assignment is a mapping that assigns 0 (false) or 1 (true) to each propositional variable. A propositional formula is satisfied by an assignment if it is true under the usual truth-functional interpretation of the connectives and the truth values assigned to the variables. Given a non-clausal MaxSAT instance ϕ, non-clausal MaxSAT is the problem of finding an assignment of ϕ that minimizes the number of unsatisfied formulas. Given a non-clausal MinSAT instance ϕ, non-clausal MinSAT is the problem of finding an assignment of ϕ that maximizes the number of unsatisfied formulas.

A weighted formula is a pair (A, w), where A is a propositional formula and w, its weight, is a positive number. Given a multiset ϕ composed of hard formulas and soft weighted formulas, non-clausal Weighted Partial MaxSAT is the problem of finding an assignment of ϕ that satisfies all the hard formulas and minimizes the sum of weights of unsatisfied soft formulas. Non-clausal Weighted Partial MinSAT is the problem of finding an assignment of ϕ that satisfies all the hard formulas and maximizes the sum of weights of unsatisfied soft formulas.

The rest of the chapter is organized as follows. Section 4.2 defines the direct, improved and Tseitin-based transformations and proves their correctness. Section 4.3 defines a tableau calculus for non-clausal MaxSAT and proves its completeness. Moreover, it describes how the tableau calculus for non-clausal MaxSAT can be used to solve non-clausal MinSAT. Section 4.4 concludes the chapter.

4.2 Clausal form transformations

There are a number of algorithms that transform a set of arbitrary propositional formulas into a satisfiability equivalent set of clauses [179, 188]. Thus, SAT solvers requiring the input in clausal form can decide the satisfiability of a set of propositional formulas by using those algorithms. Unfortunately, some clausal form transformations used in SAT are not valid in MaxSAT and MinSAT. The reason is that they do not preserve the minimum/maximum number of unsatisfied formulas in the input set of formulas. It is therefore important to analyze if existing SAT clausal form transformations are valid in MaxSAT/MinSAT, as well as to investigate how they can be adapted to deal with MaxSAT/MinSAT instances when they are not valid.

In this section, we address the problem of deriving a multiset of clauses ψ from a multiset of arbitrary propositional formulas ϕ so that the minimum/maximum number of unsatisfied clauses in ψ is equal to the minimum/maximum number of unsatisfied formulas in ϕ. Thus, by deriving such cost-preserving multisets, we provide a way of solving the MaxSAT/MinSAT problem of a multiset of arbitrary propositional formulas with MaxSAT/MinSAT solvers in which the input is required to be a multiset of clauses.

We define below three different MaxSAT/MinSAT clausal form transformations. The first transformation, called direct transformation, adds an additional step to the direct SAT clausal form transformation based on applying logical equivalences. The second transformation, called improved transformation, is a variant of the first one that has the advantage of producing more compact encodings. The third transformation, called Tseitin-based transformation, avoids the combinatorial explosion of the other transformations by introducing auxiliary variables to rename subformulas.

In the rest of the section, it is clear from the context when we refer to MaxSAT/MinSAT or non-clausal MaxSAT/MinSAT.

4.2.1 The direct MaxSAT/MinSAT clausal form transformation

For each formula in a multiset of propositional formulas ϕ, we can derive its conjunctive normal form (CNF) with the procedure below, where the uppercase letters A, B and C denote propositional formulas.

1. Remove all the occurrences of the implications using the following rules:

$$(A \to B) \rightsquigarrow (\neg A \lor B)$$

$$(A \leftrightarrow B) \rightsquigarrow (\neg A \lor B) \land (A \lor \neg B)$$

2. Reduce the scope of negation until negations appear only in front of literals using the following rules:

$$\neg\neg A \rightsquigarrow A$$

$$\neg(A \lor B) \rightsquigarrow (\neg A \land \neg B)$$

$$\neg(A \land B) \rightsquigarrow (\neg A \lor \neg B)$$

3. Derive a multiset of conjunctions of clauses using the following rules:

$$A \lor (B \land C) \rightsquigarrow (A \lor B) \land (A \lor C)$$

$$(A \land B) \lor C \rightsquigarrow (A \lor C) \land (B \lor C)$$

4. Remove clauses containing a literal and its complementary because they are tautological.

In SAT, the conjunction of clauses obtained in step 4 is usually represented by a set of clauses, where each clause is interchangeably represented by the disjunction of its literals or the set of its literals. In this thesis, we refer to such a set of clauses as *direct SAT clausal form.*

Example 4.1. Let $\phi = \{(\neg x_1 \leftrightarrow x_1) \land (\neg x_2 \leftrightarrow x_2), \neg x_1 \lor x_2\}$ be a multiset of propositional formulas, containing the formula $(\neg x_1 \leftrightarrow x_1) \land (\neg x_2 \leftrightarrow x_2)$ and the formula $\neg x_1 \lor x_2$, which is already in CNF. Applying the above procedure, we get that the CNF of the first formula is $x_1 \land \neg x_1 \land x_2 \land \neg x_2$. Thus, the SAT clausal form of ϕ is $\{\{x_1\}, \{\neg x_1\}, \{x_2\}, \{\neg x_2\}, \{\neg x_1, x_2\}\}$, which is $\{x_1, \neg x_1, x_2, \neg x_2, \neg x_1 \lor x_2\}$ when clauses are represented by disjunctions of literals instead of sets of literals.

If we look at the resulting multiset of clauses of Example 4.1 as a MaxSAT instance, then an exact MaxSAT solver will conclude that the minimum number of unsatisfied clauses in the clausal form of ϕ is 2, because we can derive a contradiction from $\{x_1\}$ and $\{\neg x_1\}$, and another from $\{x_2\}$ and $\{\neg x_2\}$. However, the minimum number of unsatisfied propositional formulas in ϕ is 1. Hence, the described SAT clausal form transformation is not valid in MaxSAT, because it preserves logical equivalence between the input multiset of propositional formulas and the resulting multiset of clauses but does not preserve the minimum number of unsatisfied formulas. In other words, it is not cost-preserving.

To overcome this drawback, we add an additional step to the previous procedure. We start by considering the CNF of each input formula as a conjunction of clauses $C_1 \land C_2 \land \cdots \land C_m$ and then impose that exactly one contradiction can be derived when at least one of such clauses is unsatisfied. To this end, we must derive a multiset of clauses from $C_1 \land C_2 \land \cdots \land C_m$ by applying the following cost-preserving rules:

R1: $A \land B \rightsquigarrow \{A, \neg A \lor B\}$.

R2: $\neg(A \lor B) \rightsquigarrow \{\neg A, A \lor \neg B\}$.

R3: $\{A, B\} \lor C \rightsquigarrow \{A \lor C, B \lor C\}$.

In the following, we refer to the multiset of clauses obtained by applying R1, R2 and R3 as *direct MaxSAT clausal form.*

We now prove the correctness of these rules. Actually, we prove that the rules preserve the number of unsatisfied formulas for each possible truth assignment. This implies that the rules preserve both the maximum and minimum number of unsatisfied formulas and, therefore, they are cost-preserving in the case of MaxSAT and in the case of MinSAT. Hence, the direct MaxSAT clausal form is also a direct MinSAT clausal form.

Proposition 4.1. Let A and B be propositional formulas. For every truth assignment of A and B, the number of unsatisfied formulas in $A \wedge B$ is equal to the number of unsatisfied formulas in $\{A, \neg A \vee B\}$.

Proof Assume that I is an assignment that unsatisfies $A \wedge B$. Then, I unsatisfies exactly one formula of $\{A, \neg A \vee B\}$ because A and $\neg A$ cannot be simultaneously unsatisfied and $\{A, \neg A \vee B\}$ is only satisfied when A and B evaluate to true. Assume now that I is an assignment that unsatisfies $\{A, \neg A \vee B\}$. Then, either I unsatisfies A or I satisfies A and unsatisfies B. In both cases, I unsatisfies $A \wedge B$. □

Proposition 4.2. Let A and B be propositional formulas. For every truth assignment of A and B, the number of unsatisfied formulas in $\neg(A \vee B)$ is equal to the number of unsatisfied formulas in $\{\neg A, A \vee \neg B\}$.

Proof Assume that I is an assignment that unsatisfies $\neg(A \vee B)$. Then, I unsatisfies exactly one formula of $\{\neg A, A \vee \neg B\}$ because A and $\neg A$ cannot be simultaneously unsatisfied and $\{\neg A, A \vee \neg B\}$ is only satisfied when A and B evaluate to false.

Assume now that I is an assignment that unsatisfies $\{\neg A, A \vee \neg B\}$. Then, either I satisfies A or I unsatisfies A and satisfies B. In both cases, I unsatisfies $\neg(A \vee B)$. □

Proposition 4.3. Let A, B and C be propositional formulas. For every truth assignment of A,B and C, the number of unsatisfied formulas in $\{A, B\} \vee C$ is equal to the number of unsatisfied formulas in $\{A \vee C, B \vee C\}$.

Proof The correctness of the rule follows from the fact that an assignment unsatisfies a disjunction iff it unsatisfies each disjunct in the disjunction. □

Example 4.2. Given the formula $(x_1 \vee x_2) \wedge (x_3 \vee x_4)$, which is already in CNF, we convert it to a cost-preserving multiset of clauses for MaxSAT and MinSAT as follows:

$$\begin{array}{l}
(x_1 \vee x_2) \wedge (x_3 \vee x_4) =^{R1} \\
\{x_1 \vee x_2, \neg(x_1 \vee x_2) \vee (x_3 \vee x_4)\} =^{R2} \\
\{x_1 \vee x_2, \{\neg x_1, x_1 \vee \neg x_2\} \vee (x_3 \vee x_4)\} =^{R3} \\
\{x_1 \vee x_2, \neg x_1 \vee x_3 \vee x_4, x_1 \vee \neg x_2 \vee x_3 \vee x_4\}
\end{array}$$

Note that the derived clausal form is larger than the direct SAT clausal form.

4.2.2 The improved MaxSAT/MinSAT clausal form transformation

A way of improving the previous transformation is by introducing auxiliary variables in the direct SAT clausal form. As a result, we obtain a more compact clausal form that is a Partial MaxSAT/MinSAT instance.

Definition 4.1. Let $\phi = \{A_1, \ldots, A_m, \ldots, A_n\}$ be a multiset of propositional formulas such that $A_1, \ldots, A_m$ are not clauses and $A_{m+1}, \ldots, A_n$ are clauses, let $CF(A_i) = \{C_1^i, \ldots, C_{r_i}^i\}$ be the multiset of clauses of the direct SAT clausal form of A_i for $i = 1, \ldots, m$, and let $y_{A_1}, \ldots, y_{A_m}$ be auxiliary propositional variables. The improved MaxSAT clausal form of ϕ is the Partial MaxSAT instance that has as hard clauses the multiset

$$\bigcup_{i=1}^{m} \{C_1^i \vee \neg y_{A_i}, \ldots, C_{r_i}^i \vee \neg y_{A_i}\} \tag{4.1}$$

and as soft clauses the multiset

$$\bigcup_{i=1}^{m} y_{A_i} \cup \bigcup_{j=m+1}^{n} A_j \,. \tag{4.2}$$

Example 4.3. Given the multiset of propositional formulas $\{x_1 \wedge (\neg x_1 \vee x_2), (x_3 \vee x_2) \wedge (\neg x_3 \vee x_2), \neg x_1 \vee \neg x_2\}$, whose formulas are in CNF, we derive the Partial MaxSAT instance that contains the following hard clauses:

$$\begin{array}{l} x_1 \vee \neg y_1 \\ \neg x_1 \vee x_2 \vee \neg y_1 \\ x_3 \vee x_2 \vee \neg y_2 \\ \neg x_3 \vee x_2 \vee \neg y_2 \end{array}$$

and the following soft clauses:

$$\begin{array}{l} y_1 \\ y_2 \\ \neg x_1 \vee \neg x_2 \end{array}$$

Observe that the the minimum number of unsatisfied clauses in the derived clausal form is 1, and coincides with the minimum number of unsatisfied formulas in the input multiset.

The following proposition states that the minimum number of propositional formulas that can be unsatisfied in the input multiset is equal to the the minimum number of soft clauses that can be unsatisfied in the resulting Partial MaxSAT instance.

Proposition 4.4. The improved MaxSAT clausal form transformation is cost-preserving.

Proof It follows from the fact that all the occurrences of the auxiliary variables y_{A_i} in the hard part of the improved MaxSAT clausal form have negative polarity. Then, when at least one clause of the direct SAT clausal form of the propositional formula A_i

is unsatisfied, y_{A_i} must be set to false to satisfy the hard part and the unit soft clause y_{A_i} becomes unsatisfied. In this way, an optimal assignment of the input multiset of propositional formulas unsatisfies A_i iff an optimal assignment of the improved MaxSAT clausal form unsatisfies the unit clause y_{A_i}, which is the single soft clauses related to A_i. □

The improved MaxSAT clausal form transformation is not valid for MinSAT. Example 4.3 provides a counterexample: the maximum number of unsatisfied clauses in the derived instance is 3, but the maximum number of unsatisfied formulas in the input multiset is 2. To overcome this problem, we need to define an improved MinSAT clausal form.

Definition 4.2. Let $\phi = \{A_1, \ldots, A_m, \ldots, A_n\}$ be a multiset of propositional formulas such that $A_1, \ldots, A_m$ are not clauses and $A_{m+1}, \ldots, A_n$ are clauses, let $CF(A_i) = \{C_1^i, \ldots, C_{r_i}^i\}$ be the multiset of clauses of the direct SAT clausal form of A_i for $i = 1, \ldots, m$, and let $y_{A_{11}}, \ldots, y_{A_{1r_1}}, \ldots, y_{A_{m1}}, \ldots, y_{A_{mr_m}}$ be auxiliary propositional variables. The improved MinSAT clausal form of ϕ is the Partial MinSAT instance that has as hard clauses the multiset

$$\bigcup_{i=1}^{m} \{\neg C_1^i \vee y_{A_{i1}}, \ldots, \neg C_{r_i}^i \vee y_{A_{ir_i}}\} \tag{4.3}$$

and as soft clauses the multiset

$$\bigcup_{i=1}^{m} y_{A_{i1}} \cup \bigcup_{i=1}^{m} \neg y_{A_{i1}} \vee y_{A_{i2}} \cup \cdots \cup \bigcup_{i=1}^{m} \neg y_{A_{i1}} \vee \neg y_{A_{i2}} \cdots \vee y_{A_{ir_i}} \cup \bigcup_{j=m+1}^{n} A_j \,. \tag{4.4}$$

The main differences between the improved MaxSAT and MinSAT clausal forms are the following: (i) for each input formula A_i, we need as many auxiliary propositional variables as clauses in the direct SAT clausal form of A_i; (ii) The clauses added in the hard part are binary; (iii) when a variable $y_{A_{ij}}$ is unsatisfied, the clause C_j^i is unsatisfied; and (iv) the soft clauses $y_{A_{i1}}, \neg y_{A_{i1}} \vee y_{A_{i2}}, \ldots, \neg y_{A_{i1}} \vee \neg y_{A_{i2}} \cdots \vee y_{A_{ir_i}}$ guarantee that exactly one of these soft clauses is unsatisfied when at least one clause in $CF(A_i) = \{C_1^i, \ldots, C_{r_i}^i\}$ is unsatisfied.

Example 4.4. Given the multiset of propositional formulas $\{x_1 \wedge (\neg x_1 \vee x_2), (x_3 \vee x_2) \wedge (\neg x_3 \vee x_2), \neg x_1 \vee \neg x_2\}$, we derive the Partial MinSAT instance that contains the following hard clauses:

$$\begin{array}{l} \neg x_1 \vee y_{11} \\ x_1 \vee y_{12} \\ \neg x_2 \vee y_{12} \\ \neg x_3 \vee y_{21} \\ \neg x_2 \vee y_{21} \\ x_3 \vee y_{22} \\ \neg x_2 \vee y_{22} \end{array}$$

and the following soft clauses:

$$
\begin{array}{l}
y_{11} \\
\neg y_{11} \vee y_{12} \\
y_{21} \\
\neg y_{21} \vee y_{22} \\
\neg x_1 \vee \neg x_2
\end{array}
$$

Observe that the maximum number of unsatisfied clauses in the derived clausal form is 2, and coincides with the maximum number of unsatisfied formulas in the input multiset.

The main problem of the two preceding transformations is that they can produce multisets of clauses whose size is exponential in the size of the corresponding input propositional formulas due to the application of the distributivity laws. To get a linear-size transformation, we should use the transformation defined in the next section.

4.2.3 The Tseitin-style MaxSAT/MinSAT clausal form transformation

The way of obtaining a clausal form from a propositional formula A with the Tseitin transformation [188] in SAT relies on adding an auxiliary variable y_ρ for each subformula ρ of A that is not a literal. Each auxiliary variable y_ρ renames a subformula ρ, depending on its top-most connective, by adding one of the following equivalences:

- $y_\rho \leftrightarrow y_B \circ y_C$ if $\rho = B \circ C$ and $\circ \in \{\wedge, \vee, \leftrightarrow, \rightarrow\}$
- $y_\rho \leftrightarrow \neg y_B$ if $\rho = \neg B$

where B and C are subformulas of ρ and y_B (y_C) is equal to B (C) if B (C) is a literal.

More precisely, given a propositional formula A that it is not a clause, the Tseitin transformation derives the following clausal form:

$$\{y_A\} \cup \left(\bigcup_{\substack{\rho \in SF(A) \\ \rho \notin Lit(A)}} Def(A, \rho) \right) \tag{4.5}$$

where y_A is the auxiliary variable associated with A, $SF(A)$ is the set of subformulas of A, $Lit(A)$ is the set of literals occurring in A, and $Def(A, \rho)$ is the definition of subformula ρ in A (see Definition 4.3).

Definition 4.3. Given a propositional formula A and a subformula ρ of A that is not a literal, the definition of subformula ρ in A, denoted by $Def(A, \rho)$, is defined as follows:

- If $\rho = B \wedge C$, then

$$Def(A, \rho) = \{\neg y_\rho \vee y_B, \neg y_\rho \vee y_C, y_\rho \vee \neg y_B \vee \neg y_C\}$$

- If $\rho = B \vee C$, then

$$Def(A,\rho) = \{\neg y_\rho \vee y_B \vee y_C, y_\rho \vee \neg y_B, y_\rho \vee \neg y_C\}$$

- If $\rho = B \rightarrow C$, then

$$Def(A,\rho) = \{\neg y_\rho \vee \neg y_B \vee y_C, y_\rho \vee y_B, y_\rho \vee \neg y_C\}$$

- If $\rho = B \leftrightarrow C$, then

$$\begin{aligned} Def(A,\rho) = \ & \{y_\rho \vee y_B \vee y_C, y_\rho \vee \neg y_B \vee \neg y_C, \\ & \neg y_\rho \vee \neg y_B \vee y_C, \neg y_\rho \vee y_B \vee \neg y_C\} \end{aligned}$$

- If $\rho = \neg B$, then

$$Def(A,\rho) = \{\neg y_\rho \vee \neg y_B, y_\rho \vee y_B\}$$

Example 4.5. The Tseitin-style SAT clausal form transformation of the propositional formula $(\neg x_1 \leftrightarrow x_1) \wedge (\neg x_2 \leftrightarrow x_2)$ of Example 4.1 is as follows:

$$\begin{aligned} &\{y_1, \\ &\neg y_1 \vee y_2, \neg y_1 \vee y_3, y_1 \vee \neg y_2 \vee \neg y_3, \\ &\neg y_2 \vee x_1, \neg y_2 \vee \neg x_1, \\ &\neg y_3 \vee x_2, \neg y_3 \vee \neg x_2\} \end{aligned}$$

where y_1 denotes $y_{(\neg x_1 \leftrightarrow x_1)\wedge(\neg x_2 \leftrightarrow x_2)}$, y_2 denotes $y_{(\neg x_1 \leftrightarrow x_1)}$ and y_3 denotes $y_{(\neg x_2 \leftrightarrow x_2)}$. Note that the second line corresponds to $y_1 \leftrightarrow y_2 \wedge y_3$, the thirth line to $y_2 \leftrightarrow (\neg x_1 \leftrightarrow x_1)$ and the fourth line to $y_3 \leftrightarrow (\neg x_2 \leftrightarrow x_2)$. Also note that tautological clauses have been removed.

The next definition provides a Tseitin-style clausal form transformation for MaxSAT.

Definition 4.4. Let $\phi = \{A_1, \ldots, A_m, \ldots, A_n\}$ be a multiset of propositional formulas such that $A_1, \ldots, A_m$ are not clauses and $A_{m+1}, \ldots, A_n$ are clauses, and let $T(A_i)$ be the multiset of clauses derived by applying Equation 4.5 for $i = 1, \ldots, m$. The Tseitin-style MaxSAT clausal form transformation of ϕ is the Partial MaxSAT instance that has as hard clauses the multiset

$$\bigcup_{i=1}^{m} (T(A_i) \setminus \{y_{A_i}\}) \tag{4.6}$$

and as soft clauses the multiset

$$\bigcup_{i=1}^{m} y_{A_i} \cup \bigcup_{j=m+1}^{n} A_j \, . \tag{4.7}$$

Example 4.6. The Tseitin-style MaxSAT clausal form transformation of $\phi = \{x_1 \wedge x_2, x_3 \wedge x_4, \neg x_1 \vee \neg x_3, \neg x_2 \vee \neg x_4\}$ is the Partial MaxSAT instance that has the following hard clauses:

$$\begin{array}{l} \neg y_1 \vee x_1 \\ \neg y_1 \vee x_2 \\ y_1 \vee \neg x_1 \vee \neg x_2 \\ \neg y_2 \vee x_3 \\ \neg y_2 \vee x_4 \\ y_2 \vee \neg x_3 \vee \neg x_4 \end{array}$$

and the following soft clauses:

$$\begin{array}{l} y_1 \\ y_2 \\ \neg x_1 \vee \neg x_3 \\ \neg x_2 \vee \neg x_4 \end{array}$$

Proposition 4.5. The Tseitin-style MaxSAT clausal form transformation is cost-preserving.

Proof It follows from the fact that we can build a satisfying assignment of $T(A_i) \setminus \{y_{A_i}\}$. When the satisfaction of $T(A_i) \setminus \{y_{A_i}\}$ forces y_{A_i} to be true, then A_i is satisfied; and when it forces y_{A_i} to be false, then A_i is unsatisfied. Therefore, the minimum number of unsatisfied soft clauses in the Tseitin-style MaxSAT clausal form is equal to the minimum number of unsatisfied formulas in the input multiset of propositional formulas. Note that the soft unit clause y_{A_i} is the single soft clause related to A_i and the variable y_{A_i} does not appear in the Tseitin transformations of the rest of formulas of the input multiset. □

Interestingly, the Tseitin-style MaxSAT clausal form transformation is also a Tseitin-style MinSAT clausal form transformation. A MaxSAT solver minimizes the variables y_{A_i} that are false and, therefore, maximizes the number of formulas A_i that are satisfied. A MinSAT solver maximizes the variables y_{A_i} that are false and, therefore, minimizes the number of formulas A_i that are satisfied. This case is interesting because the same encoding is valid for both MaxSAT and MinSAT.

4.2.4 Dealing with weights

If we associate a weight to each propositional formula, this weight can be easily incorporated into the clausal form transformations defined so far. In the case of the direct MaxSAT/MinSAT clausal form transformation, we associate the weight of the formula to each clause related to that formula; it works because at most one of such clauses can be unsatisfied in the derived MaxSAT/MinSAT instance. In the case of the improved clausal form transformations, for MaxSAT, we associate the weight of the formula A_i to the soft unit clause y_{A_i} , which is the single soft clause related to A_i. For MinSAT, we associate the weight of the formula A_i to each one of the following clauses $y_{A_{i1}}, \neg y_{A_{i1}} \vee y_{A_{i2}}, \ldots, \neg y_{A_{i1}} \vee \neg y_{A_{i2}} \cdots \vee y_{A_{ir_i}}$; it works because at most one of such clauses can be unsatisfied. In the case of the Tseitin-style MaxSAT/MinSAT clausal form transformations we associate the weight of the formula A_i to the soft unit clause y_{A_i} , which is the single soft clause related to A_i.

If we consider hard formulas, we can add as hard clauses any SAT clausal form transformation of the hard formulas. We do not need to use any MaxSAT/MinSAT clausal form transformations because hard clauses are always satisfied in any optimal solution.

Example 4.7. Given the multiset of propositional formulas $\phi = \{x_1 \wedge (\neg x_1 \vee x_2), (x_3 \vee x_2) \wedge (\neg x_3 \vee x_2), \neg x_1 \vee \neg x_2\}$ of Example 4.3, if we assign a weight of 3 to the first formula of ϕ, a weight of 2 to the second formula and a weight of 5 to the third formula, we get the improved MaxSAT clausal form consisting of the following hard clauses:

$$\begin{array}{l} x_1 \vee \neg y_1 \\ \neg x_1 \vee x_2 \vee \neg y_1 \\ x_3 \vee x_2 \vee \neg y_2 \\ \neg x_3 \vee x_2 \vee \neg y_2 \end{array}$$

and the following soft clauses:

$$\begin{array}{l} (y_1, 3) \\ (y_2, 2) \\ (\neg x_1 \vee \neg x_2, 5) \end{array}$$

Example 4.8. Given the multiset of propositional formulas $\phi = \{x_1 \wedge x_2, x_3 \wedge x_4, \neg x_1 \vee \neg x_3, \neg x_2 \vee \neg x_4\}$ of Example 4.6, if we assign a weight of 2 to the first two formulas and a weight of 5 to the last two formulas, and introduce the hard constraint $x_1 \leftrightarrow x_4$, we get the Tseitin-style MaxSAT clausal form consisting of the following hard clauses:

$$\begin{array}{l} \neg y_1 \vee x_1 \\ \neg y_1 \vee x_2 \\ y_1 \vee \neg x_1 \vee \neg x_2 \\ \neg y_2 \vee x_3 \\ \neg y_2 \vee x_4 \\ y_2 \vee \neg x_3 \vee \neg x_4 \\ y_3 \\ y_3 \vee x_1 \vee x_4 \\ y_3 \vee \neg x_1 \vee \neg x_4 \\ \neg y_3 \vee \neg x_1 \vee x_4 \\ \neg y_3 \vee x_1 \vee \neg x_4 \end{array}$$

and the following soft clauses:

$$\begin{array}{l} (y_1, 2) \\ (y_2, 2) \\ (\neg x_1 \vee \neg x_3, 5) \\ (\neg x_2 \vee \neg x_4, 5) \end{array}$$

α	α_1	α_2
$A \land B$	A	B
$\neg(A \lor B)$	$\neg A$	$\neg B$
$\neg(A \to B)$	A	$\neg B$

β	β_1	β_2
$A \lor B$	A	B
$\neg(A \land B)$	$\neg A$	$\neg B$
$A \to B$	$\neg A$	B

Table 4.1: α-formulas and β-formulas.

4.3 Tableaux for Non-Clausal MaxSAT and MinSAT

In this section, we define a complete tableau calculus for non-clausal MaxSAT and describe how this calculus can be used to solve non-clausal MinSAT.

Given a multiset of propositional formulas ϕ, we prove that the calculus is sound in the sense that if the minimum number of contradictions derived among the branches of a completed tableau for ϕ is m, then the minimum number of unsatisfied formulas in ϕ is m. We also prove that it is complete in the sense that if the minimum number of unsatisfied formulas in ϕ is m, then the minimum number of contradictions among the branches of any completed tableau for ϕ is m. Moreover, we describe how to extend the proposed calculus to deal with hard and weighted soft formulas. Finally, we describe how to solve non-clausal MinSAT with non-clausal MaxSAT tableaux.

Before presenting tableaux for non-clausal MaxSAT, we provide a brief introduction of tableaux for non-clausal SAT and show that they are unsound for MaxSAT.

4.3.1 Non-clausal SAT tableaux

One can group all propositional formulas of the form $(A \circ B)$ and $\neg(A \circ B)$, where A and B denote propositional formulas and $\circ \in \{\lor, \land, \to\}$, into two categories so that the presentation and proofs are simplified. We have those that act conjunctively, which are called α-formulas, and those that act disjunctively, which are called β-formulas. The different formulas in each category are displayed in Table 4.1. To complete a taxonomy of propositional formulas, excluding literals, we also need the propositional formulas of the form $\neg\neg A$. This notation is known as uniform notation [186].

Note that α is logically equivalent to $\alpha_1 \land \alpha_2$, β is logically equivalent to $\beta_1 \lor \beta_2$ and $\neg\neg A$ is logically equivalent to A. These equivalences are used to reduce the problem of finding a satisfying assignment of α to that of finding a satisfying assignment of both α_1 and α_2, of β to that of finding a satisfying assignment of β_1 or β_2 and of $\neg\neg A$ to that of finding a satisfying assignment of A. Thus, using the expansion rules of Table 4.2 we obtain a complete tableau calculus for non-clausal SAT.

The tableau method is used to determine the satisfiability of a given set of propositional formulas [73, 92, 186]. It starts creating an initial tableau composed of a single branch that has a node for each formula in the input set of formulas. Then, it applies the

Table 4.2: Tableau expansion rules for SAT

α	β	$\neg\neg A$	l
α_1	$\beta_1 \mid \beta_2$	A	$\neg l$
α_2			$\Box$
α-rule	β-rule	$\neg$-rule	$\Box$-rule

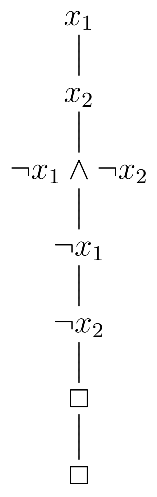


Figure 4.1: Counterexample that shows that the α-rule is unsound for non-clausal MaxSAT. The input multiset is $\phi = \{x_1, x_2, \neg x_1 \wedge \neg x_2\}$.

expansion rules of Table 4.2 until a contradiction is derived in each branch (in this case, the input set of formulas is unsatisfiable) or a branch is saturated without deriving a contradiction (in this case, the input set of formulas is satisfiable). A branch is saturated in a SAT tableau when all the possible applications of the expansion rules have been applied in that branch.

We have seen in the previous chapter that, in clausal MaxSAT tableaux, the β- and $\Box$-rule preserve the minimum number of unsatisfied clauses between a tableau and its extension; in particular, the β-rule preserves that number in at least one branch and does not decrease it in the rest of branches. So, once all branches have been saturated, the minimum number of contradictions derived among the branches of a completed tableau is the minimum number of unsatisfied clauses in the input multiset of clauses.

If we move to deal with arbitrary propositional formulas (i.e., non-clausal MaxSAT), the first problem we encounter is that the α-rule does not preserve the minimum number of unsatisfied formulas as the β-rule does for clauses. Assume that we want to solve the non-clausal MaxSAT instance $\{x_1, x_2, \neg x_1 \wedge \neg x_2\}$, whose single optimal assignment is the one that sets x_1 and x_2 to true, and only falsifies $\neg x_1 \wedge \neg x_2$. If we apply the α-rule to $\neg x_1 \wedge \neg x_2$, we add two nodes, labelled with $\neg x_1$ and $\neg x_2$, to the initial tableau. Then, we can derive two contradictions by applying the $\Box$-rule to $\{x_1, \neg x_1\}$ and $\{x_2, \neg x_2\}$, but the minimum number of formulas unsatisfied by the optimal assignment is just one. Figure 4.1 displays the resulting tableau.

The previous counterexample shows that the α-rule is unsound in MaxSAT. So, we need to define a new and sound α-rule as a first step towards getting a sound and complete non-clausal MaxSAT calculus.

4.3.2 A non-clausal MaxSAT tableau calculus

We define a non-clausal MaxSAT tableau calculus and prove its soundness and completeness. In the rest of the section, unless otherwise stated, when we say tableau we refer to a non-clausal MaxSAT tableau.

Definition 4.5. A tableau is a tree with a finite number of branches whose nodes are labelled by either a propositional formula or a box ($\Box$). A box in a tableau denotes a contradiction. A branch is a maximal path in a tree, and we assume that branches have a finite number of nodes.

Definition 4.6. Let $\phi = \{\phi_1, \ldots, \phi_m\}$ be a multiset of propositional formulas. A tableau for ϕ is constructed by a sequence of applications of the following rules:

Initialize A tree with a single branch with m nodes such that each node is labelled with a formula of ϕ is a tableau for ϕ. Such a tableau is called initial tableau and its formulas are declared active.

Given a tableau T for ϕ, a branch b of T, and a node of b labelled with an active formula F,

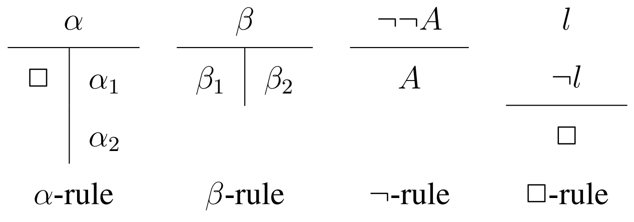
α-**rule** If F is of type α, the tableau obtained by appending a new left node below b labelled with $\Box$ and a new right branch with two nodes below b labelled with α_1 and α_2 is a tableau for ϕ. Formula F becomes inactive in b and α_1 and α_2 are declared active.

β-**rule** If F is of type β, the tableau obtained by appending a new left node below b labelled with β_1 and a new right node below b labelled with β_2 is a tableau for ϕ. Formula F becomes inactive in b and β_1 and β_2 are declared active.

$\neg$-**rule** If F is of type $\neg\neg A$, the tableau obtained by appending a new node below b labelled with A is a tableau for ϕ. Formula $\neg\neg A$ becomes inactive in b and A is declared active.

$\Box$-**rule** Given a tableau T for ϕ, a branch b of T, and two nodes of b labelled with two active complementary literals l and $\neg l$, the tableau obtained by appending a node below b labelled with $\Box$ is a tableau for ϕ. Literals l and $\neg l$ become inactive in b.

Table 4.3: Tableau expansion rules for non-clausal MaxSAT



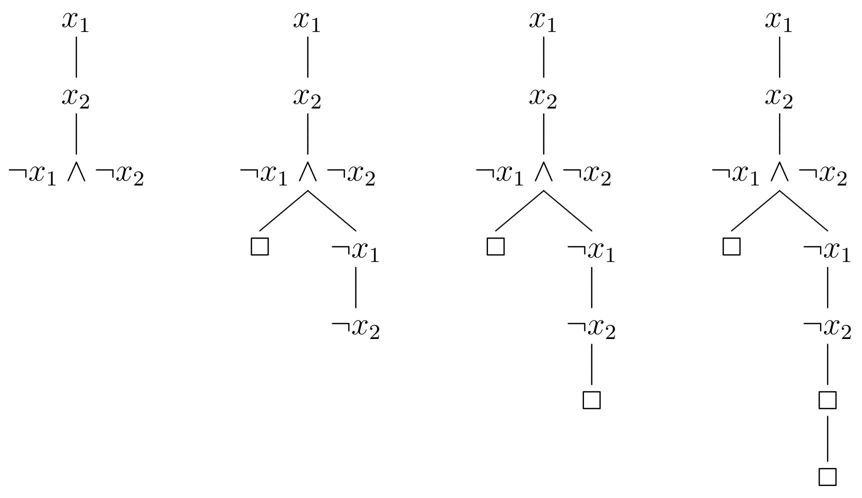


Figure 4.2: A tableau for the non-clausal MaxSAT instance $\{x_1, x_2, \neg x_1 \land \neg x_2\}$.

The expansion rules of the previous definition are summarized in Table 4.3. Note that all the rules preserve the number of premises falsified by an assignment I in at least one branch and do not decrease that number in the other branch (if any). In particular, in the α-rule, we have that if I falsifies α, the left branch contains one contradiction and α_1 and α_2 cannot be used to derive any other contradiction in that branch because they are not expanded; moreover, I falsifies α_1 or α_2 (or both) on the right branch. On the other hand, if I satisfies α, then I also satisfies α_1 and α_2 on the right branch.

Definition 4.7. Let T be a tableau for a multiset of propositional formulas ϕ. A branch b of T is saturated when no further expansion rules can be applied on b, and T is completed when all its branches are saturated. The cost of a saturated branch is the number of boxes on the branch. The cost of a completed tableau is the minimum cost among all its branches.

As we show below, the minimum number of formulas that can be unsatisfied in a multiset of propositional formulas ϕ is k iff the cost of a completed tableau for ϕ is k. Thus, the systematic construction of a completed tableau for ϕ provides an exact method for non-clausal MaxSAT.

Example 4.9. We can determine the minimum number of unsatisfied formulas in the multiset $\phi = \{x_1, x_2, \neg x_1 \wedge \neg x_2\}$ using the previous tableau calculus. Figure 4.2 displays how the tableau is constructed. We start by constructing the initial tableau (the leftmost tableau) and then apply the α-rule to $\neg x_1 \wedge \neg x_2$, getting as a result the second tableau in the figure. The leftmost branch is saturated and we apply the $\Box$-rule to $\{x_1, \neg x_1\}$ on the rightmost branch, getting as a result the third tableau. Finally, we apply the $\Box$-rule to $\{x_2, \neg x_2\}$ on the same branch and get the rightmost tableau in the figure. Since the minimum number of boxes among the branches of the last tableau is 1, the minimum number of formulas that can be unsatisfied in ϕ is 1.

4.3.3 Soundness and completeness

We prove the soundness and completeness of the proposed tableau calculus for non-clausal MaxSAT. We start by proving two propositions needed later.

Proposition 4.6. A tableau for a multiset of propositional formulas ϕ is completed in a finite number of steps.

Proof. We start by creating an initial tableau and then apply rules in the newly created branches until they are saturated. The α-, β- and $\neg$-rule reduce the number of connectives. Since we began with a finite number of connectives, these rules can only be applied a finite number of times. The $\Box$-rule inactivates two literals and adds a box. Since we began with a finite number of literals and boxes cannot be premises of any expansion rule, this rule can only be applied a finite number of times. Hence, the construction of any completed tableau terminates in a finite number of steps. $\Box$

Proposition 4.7. An assignment I falsifies k premises of a α-, β-, $\neg$- and $\Box$-rule iff assignment I falsifies k conclusions in one branch of the conclusions of the rule and at least k conclusions in the other branch (if any).

Proof. We prove the result for each rule:

- $\Box$-rule: Any assignment I always falsifies one premise and satisfies the other. Since the single conclusion is a box and denotes a contradiction, I falsifies the same number of formulas in the premises and the conclusion.
- α-rule: If I falsifies the premise of the rule, then I falsifies at least one conclusion in each branch. The left conclusion is a box and is falsified by any assignment, and I falsifies α_1 or α_2 (o both) of the right conclusion. On the other direction, if I falsifies at least one conclusion in each branch, then I falsifies α_1 or α_2 (o both) and therefore I falsifies the premise $\alpha_1 \wedge \alpha_2$.
- β-rule: If I falsifies the premise of the rule, then I falsifies β_1 and β_2, and so the left (β_1) and right (β_2) conclusions are falsified by I. On the other direction, if I falsifies both conclusions, then I falsifies $\beta_1 \vee \beta_2$.

- The $\neg$-rule: Since any assignment I that falsifies $\neg\neg A$ also falsifies A, and vice versa, I falsifies the premise iff I falsifies the conclusion.

$\square$

Theorem 4.1. Soundness & completeness. The cost of a completed tableau for a multiset of formulas ϕ is k iff the minimum number of unsatisfied formulas in ϕ is k.

Proof. (*Soundness:*) T was obtained by creating a sequence of tableaux $T_0, \ldots, T_n$ ($n \geq 0$) such that T_0 is an initial tableau for ϕ, $T_n = T$, and T_i was obtained by a single application of the α-, β-, $\neg$- or $\square$-rule on an branch of T_{i-1} for $i = 1, \ldots, n$. By Proposition 4.8, we know that such a sequence is finite. Since T has cost m, T_n contains one branch b with exactly m boxes and the rest of branches contain at least m boxes. Moreover, the active formulas in the branches of T_n are non-complementary literals; otherwise we could yet apply expansion rules and T_n could not be completed. The assignment that sets to true each active literal in b, only falsifies the m boxes and there cannot be any assignment satisfying less than m formulas in a branch of T_n because each branch contains at least m boxes. Therefore, the minimum number of active formulas than can be unsatisfied among the branches of T_n is m.

Proposition 4.7 guarantees that the minimum number of unsatisfied active formulas is preserved in the sequence of tableaux $T_0, \ldots, T_n$. Thus, the minimum number of unsatisfied active formulas in T_0 is also m. Since T_0 is formed by a single branch that only contains the formulas in ϕ and all these formulas are active, the minimum number of formulas that can be unsatisfied in ϕ is m.

(*Completeness:*) Assume that there is a completed tableau T for ϕ that does not have cost m. We distinguish two cases:

(i) T has a branch b of cost k, where $k < m$. Then, T has a branch with k boxes and a satisfiable multiset of non-complementary literals because T is completed. This implies that the minimum number of unsatisfied active formulas among the branches of T is at most k. By Proposition 4.7, this also holds for T_0, but this is in contradiction with m being the minimum number of unsatisfied formulas in ϕ because $k < m$. Thus, any branch of T has at least cost m.

(ii) T has no branch of cost m. This is in contradiction with m being the minimum number of unsatisfied formulas in ϕ. Since the tableau rules preserve the minimum number of unsatisfied formulas and the branches of any completed tableau only contain active formulas that are boxes or non-complementary literals, T must have a saturated branch with m boxes. Thus, T has a branch of cost m.

Hence, each completed tableau T for a multiset of formulas ϕ has cost m if the minimum number of formulas that can be unsatisfied in ϕ is m. $\square$

4.3.4 Extension to hard and weighted formulas

We presented the tableau calculus for non-clausal Unweighted MaxSAT (i.e.; non-clausal MaxSAT) for ease of presentation but tableaux can be extended to deal with hard and soft formulas, and soft formulas can be weighted as well.

In the case of non-clausal Partial MaxSAT, there are three basic observations:

- The hard literals of the initial tableau, as well as any other literal derived by the application of an expansion rule to an input hard formula or a subformula derived from a hard formula, always remain active. In the rest of the section, we will refer to such literals as hard literals and to the subformulas derived from a hard formula as hard subformulas.
- If the $\Box$-rule is applied to two hard literals, then the current branch is pruned. This means that we have found a contradiction among hard clauses. This corresponds to an unfeasible solution.
- When the premise of the α-rule is a hard formula or subformula, the α-rule of Table 4.2 can be used instead of the α-rule of Table 4.3. The calculus remains sound and complete but branching is reduced. This is so because hard formulas must be satisfied by any optimal assignment.

Example 4.10. Let $\phi = \mathcal{H} \cup \mathcal{S}$ be a non-clausal Partial MaxSAT instance, where $\mathcal{H}$ is the multiset of hard formulas and $\mathcal{S}$ is the multiset of soft formulas. Given the multiset of propositional formulas $\{x_1 \land x_2 \land x_3, \neg x_1, \neg x_2, \neg x_3\}$, we analyze the different tableaux obtained when we vary the formulas declared as hard and soft.

The first tableau of Figure 4.3 displays a completed tableau when all the formulas are soft; in this case $\phi = \mathcal{H} \cup \mathcal{S} = \emptyset \cup \{x_1 \land x_2 \land x_3, \neg x_1, \neg x_2, \neg x_3\}$.

The second tableau displays a completed tableau when $x_1 \land x_2 \land x_3$ is hard and the rest of formulas are soft; in this case $\phi = \mathcal{H} \cup \mathcal{S} = \{x_1 \land x_2 \land x_3\} \cup \{\neg x_1, \neg x_2, \neg x_3\}$. Notice that the input hard formulas and derived hard subformulas are in bold. We applied the α-rule of Table 4.2 because the premise is hard.

The third tableau displays a completed tableau when $\neg x_1$, $\neg x_2$ and $\neg x_3$ are hard, and $x_1 \land x_2 \land x_3$ is soft; in this case $\phi = \mathcal{H} \cup \mathcal{S} = \{\neg x_1, \neg x_2, \neg x_3\} \cup \{x_1 \land x_2 \land x_3\}$. We applied the α-rule of Table 4.3 because the premise is soft.

The fourth tableau displays a completed tableau when $x_1 \land x_2 \land x_3$ and $\neg x_1$ are hard, and $\neg x_2$ and $\neg x_3$ are soft; in this case $\phi = \mathcal{H} \cup \mathcal{S} = \{x_1 \land x_2 \land x_3, \neg x_1\} \cup \{\neg x_2, \neg x_3\}$. Notice that the single branch of the tableau is pruned as soon as the $\Box$-rule has two hard premises ($\neg x_1$ and x_1). We use a filled box to denote that there is no feasible solution.

In the first case, the minimum number of unsatisfied soft formulas is 1. In the second case, the minimum number of unsatisfied soft formulas among the assignments that satisfy the hard formulas is 3. In the third case, the minimum number of unsatisfied soft formulas among the assignments that satisfy the hard formulas is 1. In the fourth case, there is no optimal solution because the subset of hard formulas is unsatisfiable.

Table 4.4 displays the expansion rules for weighted formulas. The α-, β- and $\neg$-rule have just one premise and the weight associated with the premise is transferred to the conclusions. The $\Box$-rule has two premises and so the contradiction takes as weight the minimum of the weights associated with the premises ($\min(w_1, w_2)$). If the premises

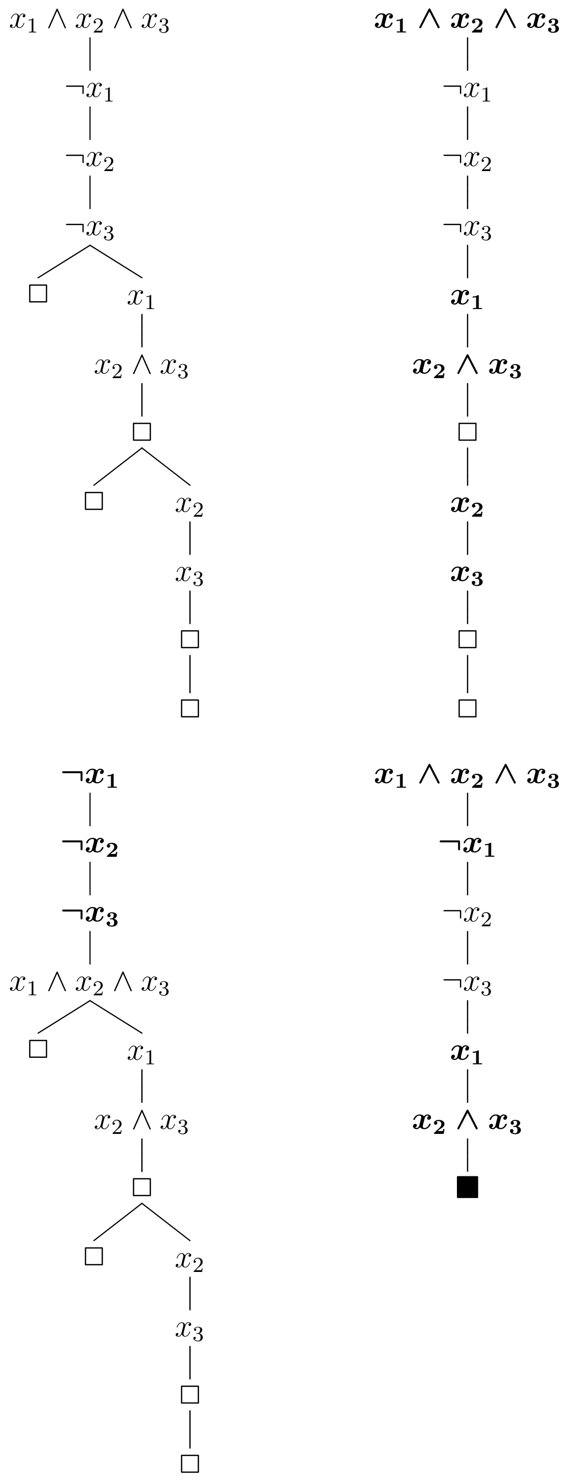


Figure 4.3: Examples of non-clausal Partial MaxSAT tableaux. Input hard formulas and derived hard subformulas are in bold.

Table 4.4: Tableau expansion rules for non-clausal Weighted MaxSAT

(α, w)	(β, w)	$(\neg\neg A, w)$	(l, w_1) $(\neg l, w_2)$
$(\Box, w) \mid (\alpha_1, w)$ $ \mid (\alpha_2, w)$	$(\beta_1, w) \mid (\beta_2, w)$	(A, w)	$(\Box, \min(w_1, w_2))$ $(l, w_1 - \min(w_1, w_2))$ $(\neg l, w_2 - \min(w_1, w_2))$
α-rule	β-rule	$\neg$-rule	$\Box$-rule

have different weights, the remaining weight in the premise with the greatest weight can be used to detect further contradictions. The compensation weight of the other premise is 0, and formulas with weight 0 are removed. In the weighted case, when a branch has repeated occurrences of a formula A, say $(A, w_1), \ldots, (A, w_s)$, such occurrences can be replaced with the single formula $(A, w_1 + \cdots + w_s)$. Moreover, the cost of a saturated weighted branch is the sum of weights of the boxes that appear in the branch, and the cost of a completed weighted tableau is the minimum cost among all its branches.

The expansion rules of Table 4.4 provide a sound and complete calculus for non-clausal Weighted MaxSAT. The correctness of such rules follows from the correctness of the unweighted tableau rules and the fact that having a weighted formula (A, w) is equivalent to having w copies of the unweighted formula A.

Example 4.11. Let $\phi = \{(\neg x_1 \rightarrow x_2, 3), (x_1 \land x_3, 2), (\neg x_1, 5), (\neg x_1, 5), (\neg x_3, 2)\}$ be a non-clausal Weighted MaxSAT instance. Figure 4.4 displays a completed tableau for ϕ. This tableau has been obtained by applying the expansion rules of Table 4.4. The costs of the branches, from left to right, are 5, 7, 5 and 7. So, the minimum sum of weights of unsatisfied formulas is 5.

Finally, we show how to solve non-clausal Weighted Partial MaxSAT instances with tableaux. The first observation is that hard formulas can be considered as weighted formulas with infinity weight, and this observation is important to understand the $\Box$-rule in Weighted Partial MaxSAT. Notice that the $\Box$-rule is the only rule with two premises; in the rest os cases, if the premise is hard, we proceed as in Partial MaxSAT, and if it is soft, we proceed as in Weighted MaxSAT. If the two premises of the $\Box$-rule are hard, then the branch is pruned because we are in front of an unfeasible solution. If the two premises are soft, then the $\Box$-rule of Table 4.4 is applied. If there is a hard premise l and a soft premise $(\neg l, w)$, then $(\Box, w)$ is derived, $(\neg l, w)$ becomes inactive and l remains active.

Example 4.12. Let $\phi = \{(x_1 \land x_3, (\neg x_1 \rightarrow x_2, 3), (\neg x_1, 5), (\neg x_2, 1), (\neg x_3, 2)\}$ be a non-clausal Weighted Partial MaxSAT instance, where the first formula is hard and the rest of formulas are soft. Figure 4.5 displays a completed tableau for ϕ. This tableau has been obtained by applying the expansion rules for non-clausal Weighted Partial

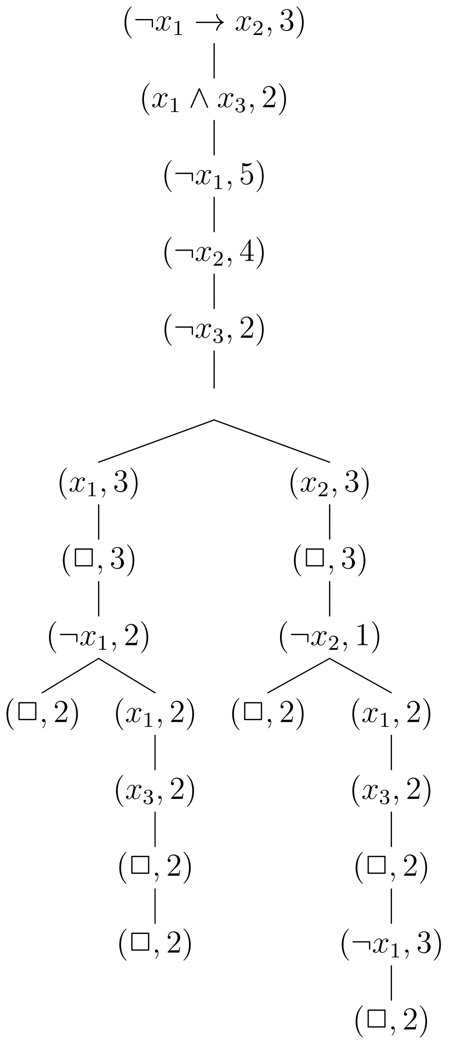


Figure 4.4: Example of a non-clausal Weighted MaxSAT tableau.

MaxSAT explained above. The cost of the left branch is 10 and the cost of the right branch is 8. Thus, the minimum sum of weights of unsatisfied soft formulas among the assignments that satisfy the hard formula is 8.

4.3.5 A tableau approach for non-clausal MinSAT

We first give a proposition and then show how non-clausal tableaux for MaxSAT can be used to solve non-clausal MinSAT.

Proposition 4.8.

Any optimal MaxSAT assignment of $\phi = \{\neg\phi_1, \ldots, \neg\phi_m\}$ is an optimal MinSAT assignment of $\phi' = \{\phi_1, \ldots, \phi_m\}$.

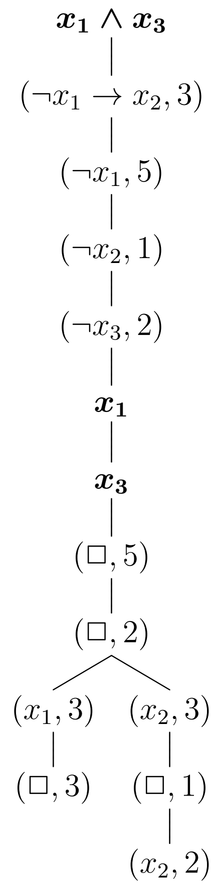


Figure 4.5: Example of a non-clausal Weighted Partial MaxSAT tableau. Input hard formulas and derived hard subformulas are in bold.

Proof. Let I be an optimal MaxSAT assignment of $\phi = \{\neg\phi_1, \ldots, \neg\phi_m\}$ and let k the number of formulas of ϕ that I unsatisfies. Then, I satisfies k formulas of $\phi' = \{\phi_1, \ldots, \phi_m\}$ because, for $i = 1, \ldots, n$, an assignment satisfies ϕ_i iff it unsatisfies $\neg\phi_i$. Moreover, there is no assignment of ϕ' that satisfies k' formulas of ϕ', where $k' < k$, because such an assignment should unsatisfy k' formulas of ϕ and this is in contradiction with the fact that I is an optimal MaxSAT assignment. Therefore, the minimum number of clauses that can be satisfied in ϕ' is k. □

It follows from the previous proof that if an optimal MaxSAT assignment I of $\phi = \{\neg\phi_1, \ldots, \neg\phi_m\}$ unsatisfies k formulas, then I is an optimal MinSAT assignment of $\phi' = \{\phi_1, \ldots, \phi_m\}$ that satisfies k formulas. Therefore, the maximum number of formulas that can be unsatisfied in ϕ' is $m - k$.

To find an optimal MinSAT solution of $\phi' = \{\phi_1, \ldots, \phi_m\}$, we must build a completed non-clausal MaxSAT tableau of $\phi = \{\neg\phi_1, \ldots, \neg\phi_m\}$. If the cost of the non-clausal MaxSAT tableau is k, the maximum number of formulas that can be unsatisfied in ϕ' is $m - k$.

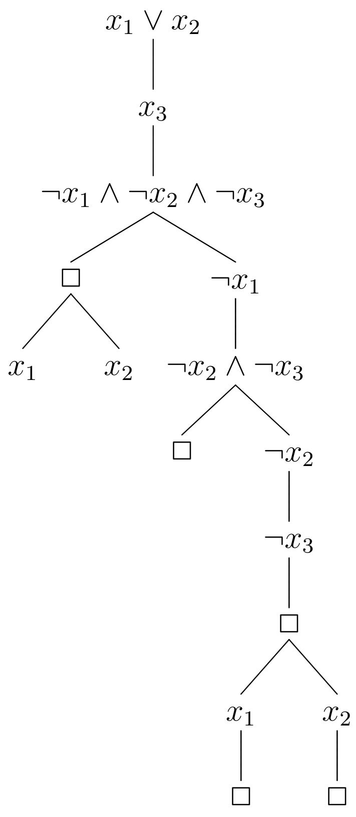


Figure 4.6: A tableau for the non-clausal MinSAT instance $\phi = \{\neg x_1 \wedge \neg x_2, \neg x_3, x_1 \vee x_2, \vee x_2\}$.

Example 4.13. Let $\phi = \{\neg x_1 \wedge \neg x_2, \neg x_3, x_1 \vee x_2, \vee x_2\}$ be a non-clausal MinSAT instance. Figure 4.6 displays a completed non-clausal MaxSAT tableau for $\{x_1 \vee x_2, x_3, \neg(x_1 \vee x_2 \vee x_3)\}$. Since the cost of the non-clausal MaxSAT tableau is 1, the maximum number of clauses that can be unsatisfied in ϕ is $3 - 1 = 2$.

If $\phi' = \{(\phi_1, w_1), \ldots, (\phi_m, w_m)\}$ is a Weighted MinSAT instance, then we must build a completed non-clausal MaxSAT tableau for

$$\phi = \{(\neg\phi_1, w_1), \ldots, (\neg\phi_m, w_m)\}.$$

.

If $\phi' = \{h_1, \ldots, h_n, (\phi_1, w_1), \ldots, (\phi_m, w_m)\}$ is a Weighted Partial MinSAT instance, then we must build a completed non-clausal MaxSAT tableau for

$$\phi = \{h_1, \ldots, h_n, (\neg\phi_1, w_1), \ldots, (\neg\phi_m, w_m)\}.$$

.

4.4 Concluding remarks

We have defined two methods for solving non-clausal MaxSAT and non-clausal MinSAT. First, we have defined the direct, improved and Tseitin-based transformations. These transformations reduce non-clausal MaxSAT/MinSAT to clausal MaxSAT/MinSAT so that modern MaxSAT/MinSAT solvers can be used to solve the MaxSAT/MinSAT problem of multisets of arbitrary propositional formulas. Second, we have defined a complete tableau calculus for non-clausal MaxSAT, which can be extended to deal with hard and soft formulas. Moreover, we have described how the tableau calculus for solving non-clausal MaxSAT can be used to solve non-clausal MinSAT.

The contributions of this chapter have inspired a tableau calculus for solving the MaxSAT problem of any finitely-valued Łukasiewicz Logic [144].

Chapter 5

Solving the Team Composition Problem in a Classroom

This chapter defines the Team Composition Problem in a Classroom (TCPC), proves that it is NP-hard, and defines two different MaxSAT models of the problem, called maximizing and minimizing encodings. It also reports on the results of an empirical investigation that shows that solving the TCPC as a MaxSAT problem is promising, and provides evidence that the minimizing encoding outperforms the maximizing encoding. Finally, it describes how the proposed approach can be extended to richer team formation problems, using the Synergistic Team Composition Model (STCM) problem as a case study.

The presentation closely follows the work published in [159]. A workshop version of this paper with preliminary results appeared in [158].

5.1 Introduction

Given a classroom containing a fixed number of students and tables, which can be of different capacities, and a list of preferred classmates to sit with for each student, the Team Composition Problem in a Classroom (TCPC) is the problem of finding an assignment of students to tables so that the preferences of the students are maximally-satisfied.

We propose to solve the TCPC as a MaxSAT problem. More precisely, given a TCPC instance ϕ, we first define a suitable MaxSAT encoding for ϕ, then find an optimal assignment for the resulting encoding with an off-the-shelf MaxSAT solver, and finally derive an optimal solution for ϕ from the optimal assignment. It is a declarative approach, because we define a model and an optimal solution is derived automatically. Furthermore, the approach is highly efficient because we may take advantage of the extremely efficient MaxSAT solvers which are publicly available.

One could think that designing an algorithm to work directly on the original problem encoding should outperform approaches that require a translation via a generic intermediate formalism such as a CSP, SAT, SMT, MaxSAT, or MinSAT. However, this line of reasoning ignores that generic solvers can benefit from many years of development by a broad research community. It is not easy to replicate this kind of effort in other domains.

The motivation behind this work is twofold. First, we aim to solve a problem posed by the director of studies of a secondary school in the area of Barcelona, though similar team formation problems are frequent in a wide range of situations and institutions. Second, we introduce a new MaxSAT application that reinforces the idea that MaxSAT is a competitive approach for solving challenging optimization problems. Moreover, we provide new benchmarks for empirically evaluating MaxSAT solvers.

In this chapter, we first define the TCPC, prove that it is NP-hard, and define two different MaxSAT models of the problem, called maximizing and minimizing encodings. Then, we report on the results of an empirical investigation that shows that solving the TCPC as a MaxSAT problem is promising, and provides evidence that the minimizing encoding outperforms the maximizing encoding. Finally, we describe how we could take into account other factors relevant to the performance of a team such as personality, expertise, competence, competitiveness, and human formation [12, 18]. To illustrate this point, we describe how the Synergistic Team Composition Model (STCM) [18] problem can be solved as a MaxSAT problem.

All the contributions of this chapter are presented for MaxSAT. Nevertheless, it is worth pointing out that the maximizing and minimizing MaxSAT encodings of the TCPC can be easily transformed into MinSAT encodings. We could then solve the TCPC instances with a MinSAT solver.

The rest of the chapter is organized as follows. Section 5.2 defines the TCPC and proves that it is NP-hard. Section 5.3 defines the maximizing and minimizing MaxSAT encodings. Section 5.4 reports on the empirical investigation. Section 5.5 illustrates how to solve the STCM problem with MaxSAT technology. Section 5.6 discuss some related work. Section 5.7 concludes the chapter.

5.2 The team composition problem in a classroom

Depending on the activity to be performed in a classroom at a given moment, the distribution of the students may need to be different. In the general case, we consider there is a fixed number of students and there is a list of preferred classmates to sit with for each student. Then, the goal is to partition students into teams, which may have different sizes, so that the preferences of the students are maximally-satisfied.

The version of the TCPC that we use as a case study in this chapter has the following constraints:

- The classroom has n students.
- The classroom has tables of 2 and 3 students with a combined capacity for n students.
- Each student has provided a list of classmates she would prefer to sit with.

The objective is to find an assignment of students to tables so that the preferences of students are maximally-satisfied. Notice that the first two constraints are hard whereas the last one is soft. We will say that a solution is *fully-satisfied* if, and only if, all the students in the same table have the rest of the students of the table in their preference list. We will say that a solution is *maximally-satisfied* if, and only if, the number of students who have their preferences satisfied is maximized. Note that a fully-satisfied solution is also a maximally-satisfied solution.

Proposition 5.1. Given n students, a classroom that has tables of 2 and 3 students with a combined capacity for n students, and a list of preferred classmates to sit with for each student, the problem of deciding if there is a fully-satisfied solution is NP-complete.

Proof. This problem belongs to NP: we can check, in polynomial time, whether or not an assignment of students to tables is a fully-satisfied solution by inspecting the lists of preferences of the students.

We now prove that this problem is NP-hard by reducing the *Partition into Triangles* (PIT) problem to it. Assume that $G = (V, E)$ is a graph, where V is the set of vertices and verifies that $|V| = 3q$ for some integer q, and E is the set of edges. The PIT problem for G is to find a partition $\{V_1, \ldots, V_q\}$ of V so that each $V_i = \{u_i, v_i.w_i\}$, $1 \leq i \leq q$, contains exactly 3 vertices and the edges $\{u_i, v_i\}$, $\{u_i, w_i\}$ and $\{v_i, w_i\}$ belong to E. This problem is NP-complete [88].

The PIT problem for G can be reduced to an instance of our problem by considering a classroom with $3q$ students, 0 tables of 2 and q tables of 3, and establishing, for each edge $\{u, v\}$ of G, a preference of student u for student v and a preference of student v for student u. Note that this reduction takes polynomial time. Then, the PIT problem has a solution if, and only if, all the students in the classroom can be sat in such a way that all the preferences of students are fully-satisfied.

□

Corollary 5.1. The TCPC is NP-hard.

Proof. This follows from the fact that every fully-satisfied solution is also a solution that is maximally-satisfied. □

We can find a fully-satisfied solution with a decision algorithm but need an optimization algorithm to find a maximally-satisfied solution. Indeed, finding a maximally-satisfied solution is in general harder than finding a fully-satisfied solution. For example, if we assume that there are just tables of 2 students, finding a fully-satisfied solution can be solved in polynomial time but finding a maximally-satisfied solution remains NP-hard.

5.3 MaxSAT encodings for the TCPC

We present two different ways of encoding the TCPC in the Weighted Partial MaxSAT formalism. In the first approach, the objective is to maximize the quality of the solution and we refer to it as the maximizing encoding. In the second approach, the objective is to minimize the quality loss and we refer to it as the minimizing encoding.

5.3.1 The maximizing encoding

We first present how the TCPC can be represented as a Weighted Partial MaxSAT instance using the maximizing encoding. To illustrate how to model the problem, we will consider that the classroom has 28 students and there are 8 tables of 2 students and 4 tables of 3 students. This is a typical classroom distribution in some secondary schools.

First of all, we define the set of Boolean variables of the encoding:

$$\{x_{ij} | 1 \leq i < j \leq 28\} \cup \{x_{ijk} | 1 \leq i < j < k \leq 28\} \cup \{y_i | 1 \leq i \leq 28\}$$

These variables have the following intended meaning: x_{ij} is true iff students i and j sit together in a table of 2; x_{ijk} is true iff students i, j and k sit together in a table of 3; and y_i is true if student i sits in a table of 2 and is false if student i sits in a table of 3. Using these variables, we create a Weighted Partial MaxSAT instance that encodes the constraints of the problem.

The encoding has the following hard clauses:

1. For each student i, where $1 \leq i \leq 28$, the encoding contains a set of hard clauses that encode the following cardinality constraint:

   (a) If $i = 1$, then
   $$\sum_{j=2}^{28} x_{1j} + \sum_{j=2}^{27} \sum_{k=j+1}^{28} x_{1jk} = 1$$

   (b) If $2 \leq i \leq 27$, then
   $$\sum_{j=1}^{i-1} x_{ji} + \sum_{j=i+1}^{28} x_{ij} + \sum_{k=2}^{i-1} \sum_{j=1}^{k-1} x_{jki} + \sum_{j=1}^{i-1} \sum_{k=i+1}^{28} x_{jik} + \sum_{j=i+1}^{27} \sum_{k=j+1}^{28} x_{ijk} = 1$$

   (c) If $i = 28$, then
   $$\sum_{j=1}^{27} x_{j28} + \sum_{j=1}^{26} \sum_{k=j+1}^{27} x_{jk28} = 1$$

   This cardinality constraint states that student i sits exactly in one table, and the table is of 2 or 3 students.

2. For each variable x_{ij}, the encoding contains the hard clauses $\neg x_{ij} \vee y_i$ and $\neg x_{ij} \vee y_j$. Note that $(\neg x_{ij} \vee y_i) \wedge (\neg x_{ij} \vee y_j)$ is equivalent to $x_{ij} \rightarrow y_i \wedge y_j$. This clause states that if x_{ij} is true, then students i and j sit in a table of 2.

3. For each variable x_{ijk}, the encoding contains the hard clauses $\neg x_{ijk} \vee \neg y_i$, $\neg x_{ijk} \vee \neg y_j$ and $\neg x_{ijk} \vee \neg y_k$. Note that $(\neg x_{ijk} \vee \neg y_i) \wedge (\neg x_{ijk} \vee \neg y_j) \wedge (\neg x_{ijk} \vee \neg y_k)$ is equivalent to $x_{ijk} \rightarrow \neg y_i \wedge \neg y_j \wedge \neg y_k$. This clause states that if x_{ijk} is true, then students i, j and k sit in a table of 3.

4. The encoding contains a set of hard clauses that encode the following cardinality constraints: $\sum_{i=1}^{28} y_i = 16$ and $\sum_{i=1}^{28} \neg y_i = 12$. These cardinality constraints state that there are 16 students sitting in tables of 2 and 12 students sitting in tables of 3.

   In practice, it is sufficient to add either the constraint $\sum_{i=1}^{28} y_i = 16$ or the constraint $\sum_{i=1}^{28} \neg y_i = 12$ because if there are exactly 16 (12) variables y_i, $1 \leq i \leq 28$, that evaluate to true (false), then the remaining 12 (16) variables must evaluate to false.

The encoding of a cardinality constraint of the form $x_1 + \ldots + x_n = k$ has $\mathcal{O}(n)$ clauses if one uses the encoding based on counters and defined in [185]. Other efficient encodings of cardinality constraints are described and analyzed in [1, 59]. In our empirical investigation, we encode the previous cardinality constraints using PBLib[1], which is a C++ tool for efficiently encoding pseudo-Boolean constraints to CNF.

Since we considered two types of tables, we just need one variable y_i for each student. If we consider n different types, then we need $\lceil \log_2 n \rceil$ variables for each student. For example, for four different types, we need two variables (y_i, y_i') and each type is represented by one of the following conjunctions: $y_i \wedge y_i'$, $\neg y_i \wedge y_i'$, $y_i \wedge \neg y_i'$ and $\neg y_i \wedge \neg y_i'$.

The soft clauses of our encoding are the following weighted unit clauses:

1. For each variable x_{ij}, $1 \leq i < j \leq 28$, the encoding contains the weighted unit clause (x_{ij}, w_{ij}).

2. For each variable x_{ijk}, $1 \leq i < j < k \leq 28$, the encoding contains the weighted unit clause (x_{ijk}, w_{ijk}).

A key aspect of our encoding is how weights are assigned to the variables of the form x_{ij} and x_{ijk}. First of all, we build a directed graph $G = (V, E)$, where V contains a vertex i for each student i in the classroom, and E contains an edge (i, j) if student i wants to sit with student j. The weight associated with each student i in G, denoted by $w(i)$, is the out-degree of the vertex i of G.[2] The weight associated with the variable x_{ij}, denoted by w_{ij}, is $2(w(i) \times w(j))$, where $w(i)$ and $w(j)$ are the weights associated with vertices i and j, respectively, in the subgraph of G induced by the set of vertices $\{i, j\}$ (i.e.; the weight of student i and j in $G(\{i, j\})$). The weight associated with the variable x_{ijk}, denoted by w_{ijk}, is $3(w(i) \times w(j) \times w(k)/8)$, where $w(i)$, $w(j)$ and $w(k)$ are the weights associated with vertices i, j and k, respectively, in $G(\{i, j, k\})$. The value of $w(i) \times w(j)$ ranges from 0 to 1 and the value of $w(i) \times w(j) \times w(k)$ ranges from 0 to 8. This explains the fact that $w(i) \times w(j) \times w(k)$ is divided by 8. Moreover, we multiply the weights by 2 in the tables of 2 and by 3 in the tables of 3. In this way, we maximize the number of satisfied students. Note that if the weight assigned to x_{ij} is 2, there are 2 satisfied students if they sit together in a table of 2, whereas if the weight assigned to x_{ijk} is 3, there are 3 satisfied students if they sit together in a table of 3. The weight w_{ij} (w_{ijk}) associated with a table of 2 (3) indicates the quality of the assignment of students

[1]http://tools.computational-logic.org/content/pblib.php

[2]The out-degree of a vertex in a directed graph is the number of edges going out of the vertex.

i and j (i, j and k) to a table of 2 (3). The bigger the weight, the better the assignment of students to tables.[3]

In the maximizing encoding, if the weight associated with a variable is 0, then the negation of this variable is added as a unit clause in the hard part. Moreover, an optimal solution corresponds to a fully-satisfied solution if, and only if, all the satisfied soft clauses of the form (x_{ij}, w_{ij}) and (x_{ijk}, w_{ijk}) have weight 2 and 3, respectively.

For fully-satisfied instances, if we add to the hard part the negation of x_{ij} (i.e., the unit hard clause $\neg x_{ij}$) for each variable x_{ij} whose associated weight is different from 2 and the negation of x_{ijk} (i.e., the unit hard clause $\neg x_{ijk}$) for each variable x_{ijk} whose associated weight is different from 3, then we do not need to add any soft clause. Moreover, any satisfying assignment of the hard part allows us to derive a fully-satisfied solution. This case can be solved either with a SAT solver or with a MaxSAT solver fed with a MaxSAT instance that only contains hard clauses. Actually, finding a fully-satisfied solution is a decision problem.

If there is no fully-satisfied solution, the problem becomes an optimization problem and the objective is to find a solution that satisfies students as much as possible. Because of that, in the general case, we add the clauses (x_{ij}, w_{ij}) and (x_{ijk}, w_{ijk}) such that $w_{ij} \neq 0$ and $w_{ijk} \neq 0$ in the soft part of the encoding. In this way, we provide a solution that maximizes the number of satisfied students. In this case, we say that we have a maximally-satisfied solution.

An optimal solution to the TCPC is obtained from a MaxSAT optimal interpretation by assigning students i and j to the same table of 2 if, and only if, the literal x_{ij} is satisfied by the optimal interpretation; and by assigning students i, j and k to the same table of 3 if, and only if, the literal x_{ijk} is satisfied by the optimal interpretation.

If an optimal interpretation satisfies the soft clause (x_{ij}, w_{ij}), then this interpretation falsifies all the soft clauses (x_{lm}, w_{lm}) and (x_{lmn}, w_{lmn}) such that l, m or n are equal to i or j because of the cardinality constraint that states that every student sits exactly in one table. A similar situation happens when the satisfied clause is of the form (x_{ijk}, w_{ijk}), corresponding to a table of 3. Thus, the number of falsified soft clauses is usually greater than the number of satisfied soft clauses, and the maximum sum of weights of satisfied clauses indicates the maximum quality that can be reached taking into account the preferences of the students.

5.3.2 The minimizing encoding

The minimizing encoding focus on minimizing the quality loss instead of maximizing the quality of the solution as in the maximizing encoding. Thus, the challenge now is to adequately represent the notion of quality loss in the TCPC and derive a more efficient encoding.

The minimizing encoding is defined over the same set of Boolean variables and has the hard constraints of the maximizing encoding. The soft clauses are derived from the

[3]Since most of the MaxSAT solvers deal with weights that are positive integers, in the experiments we multiply the weights by 100 and take the integer part.

soft clauses of the maximizing encoding as follows:

1. each soft clause (x_{ij}, w_{ij}) is replaced with the soft clause $(\neg x_{ij}, w_{max} - w_{ij})$, and

2. each soft clause (x_{ijk}, w_{ijk}) is replaced with the soft clause $(\neg x_{ijk}, w'_{max} - w_{ijk})$,

where w_{max} is the maximum weight that can be assigned to a table of 2 and w'_{max} is the maximum weight that can be assigned to a table of 3. In our encoding, $w_{max} = 2$ and $w'_{max} = 3$.

An optimal solution to the TCPC is obtained from a MaxSAT optimal interpretation by assigning students i and j to the same table of 2 if, and only if, the literal $\neg x_{ij}$ is falsified by the optimal interpretation; and by assigning students i, j and k to the same table of 3 if, and only if, the literal $\neg x_{ijk}$ is falsified by the optimal interpretation. Note that $\neg x_{ij}$ and $\neg x_{ijk}$ are falsified if, and only if, x_{ij} and x_{ijk} are satisfied. If an optimal interpretation falsifies the soft clause $(\neg x_{ij}, w'_{ij})$, then it satisfies all the soft clauses $(\neg x_{lm}, w'_{lm})$ and $(\neg x_{lmn}, w'_{lmn})$ such that l, m or n are equal to i or j because of the cardinality constraint that states that every student sits exactly in one table. A similar situation happens when the falsified clause is of the form $(\neg x_{ijk}, w'_{ijk})$.

In contrast to the maximizing encoding, the number of satisfied soft clauses in an optimal solution of the minimizing encoding is usually greater than the number of falsified soft clauses. This implies that the number of conflicts that a MaxSAT solver has to identify for finding an optimal solution is greater in the maximizing encoding than in the minimizing encoding and, as we will see in the experimental results, this has a tremendous impact on the performance of the solver.

The weight of the soft clause $(\neg x_{ij}, w_{max} - w_{ij})$ $((\neg x_{ijk}, w'_{max} - w_{ijk}))$ indicates the quality loss if students i and j (i, j and k) sit together in a table of 2 (3): the smaller the weight, the better the assignment of students to tables. In fact, the weight $w_{max} - w_{ij}$ $(w'_{max} - w_{ijk})$ is the penalty to be paid by students i and j (i, j and k) if they sit in the same table. Thus, the minimum sum of weights of falsified clauses indicates the minimum quality loss that can be reached taking into account the preferences of the students.

If the minimum sum of weights of falsified clauses in an optimal solution is 0, then this solution is fully-satisfied. Note that the clauses of the form $(\neg x_{ij}, 0)$ correspond to tables of 2 in which students i and j prefer to sit together, and the clauses of the form $(\neg x_{ijk}, 0)$ correspond to tables of 3 in which students i, j and k prefer to sit together. In practice, the clauses $(\neg x_{ij}, 0)$ and $(\neg x_{ijk}, 0)$ can be removed from the soft part and the encoding remains correct.

It is worth pointing out that the minimization approach proposed here can be extended to other combinatorial optimization problems. It is particularly useful when the resulting MaxSAT encoding has subsets of soft unit clauses whose literals appear in hard cardinality constraints. In this case, the encoding can considerably reduce the number of conflicts needed to find an optimal solution. The main difficulty of the minimizing encoding is to define a suitable weighting function that preserves the optimal solutions between the maximizing and the minimizing encodings.

Finally, it is worth mentioning that it is possible to define the previous MaxSAT encodings of the TCPC using the set of propositional variables $\{x_i^t | 1 \leq i \leq 28, 1 \leq t \leq 12\}$, where the intended meaning of x_i^t is that x_i^t is true if, and only if, student i sits at table t. However, all the experiments performed with encodings using this set of variables did not outperform the experiments performed with the encodings proposed in this section.

5.4 Experimental results

We conducted an experimental investigation to evaluate the proposed MaxSAT-based approach to the TCPC on both fully-satisfied and maximally-satisfied instances, and compared the performance of the maximizing and minimizing encodings on the selected instances. In the experiments, in order to analyze the scaling behavior, we considered different classroom capacities: the rows always have 2 tables of 2 and 1 table of 3, and the number of rows ranges from 1 to 18. Hence, the number of students per classroom ranges from 7 to 126. Besides, we assumed that each student gives a list of students she would like to sit with. We generated the preferences at random, guaranteeing that the generated instances have either fully-satisfied or maximally-satisfied solutions. We generated 50 different TCPC instances for each classroom capacity, encoded them to Weighted Partial MaxSAT, and solved the resulting maximizing and minimizing encodings with the exact MaxSAT solver WPM3 [27] using a cutoff time of 900 seconds. All the experiments were performed in a 3.60GHz Intel(R) i7-4790 with 8GB RAM.

We selected WPM3 because it was the best performing solver on our instances in preliminary tests. WPM3 reformulates the MaxSAT optimization problem into a sequence of SAT decision problems and introduces Pseudo-Boolean (PB) constraints to refine the lower bound after each execution of the SAT solver. To identify the most suitable PB constraints, WPM3 analyzes the unsatisfiable cores retrieved from the previous SAT executions.

Table 5.1 compares the maximizing and minimizing encodings on fully-satisfied instances. We observe that the minimizing encoding clearly outperforms the maximizing encoding: the minimizing encoding needs a mean time of less than one second to solve an instance, independently of the classroom capacity, but the maximizing encoding only solves all the selected instances within the cutoff time if the number of students is less than or equal to 49. The maximizing encoding only solves 43, 31 and 5 instances out of 50 when the number of students is 63, 77 and 84, respectively. It was not able to solve any instance for more than 84 students.

Table 5.2 compares the maximizing and minimizing encodings on maximally-satisfied instances. We observe that the minimizing encoding scales much better than the maximizing encoding, and also needs less time to solve an instance. While the minimizing encoding solves all the instances with less than 98 students, the maximizing encoding fails to solve some instances when there are 70 or more students. The different perfor-

Table 5.1: Experimental results for fully-satisfied instances. Students: number of students; Hard: mean number of hard clauses per instance; Variables: mean number of variables per instance; Soft _Max: mean number of soft clauses per instance in the maximizing encoding; Soft _Min: mean number of soft clauses per instance in the minimizing encoding; Time_Max: mean time, in seconds, needed to solve an instance with the maximizing encoding; and Time_Min: mean time, in seconds, needed to solve an instance with the minimizing encoding. The number of solved instances, within a cutoff time of 900s, is shown in parentheses.

Students	Hard	Variables	Soft_Max	Soft_Min	Time_Max	Time_Min
7	246	117	21	11	0,01 (50)	0,01 (50)
14	1040	659	56	35	0,01 (50)	0,01 (50)
21	2594	1916	93	60	0,05 (50)	0,01 (50)
28	5214	4239	128	85	0,25 (50)	0,01 (50)
35	9127	7937	150	98	0,63 (50)	0,01 (50)
42	14934	13387	189	124	3,65 (50)	0,01 (50)
49	22772	20889	221	147	10 (50)	0,02 (50)
56	33069	30842	257	172	63 (49)	0,03 (50)
63	46079	43552	286	191	123 (43)	0,04 (50)
70	62232	59365	324	218	191 (31)	0,06 (50)
77	81833	78584	357	240	234 (25)	0,08 (50)
84	105200	101588	386	260	286 (20)	0,11 (50)
91	132775	128730	426	286	513 (10)	0,21 (50)
98	164741	160326	456	308	634 (5)	0,25 (50)
105	201629	196844	493	332	815 (2)	0,28 (50)
112	243565	238390	522	351	0 (0)	0,30 (50)
119	291154	285487	568	385	0 (0)	0,49 (50)
126	344370	338356	590	398	0 (0)	0,62 (50)

Table 5.2: Experimental results for maximally-satisfied instances. Students: number of students; Hard: mean number of hard clauses per instance; Variables: mean number of variables per instance; Soft _Max: mean number of soft clauses per instance in the maximizing encoding; Soft _Min: mean number of soft clauses per instance in the minimizing encoding; Time_Max: mean time, in seconds, needed to solve an instance with the maximizing encoding; and Time_Min: mean time, in seconds, needed to solve an instance with the minimizing encoding. The number of solved instances, within a cutoff time of 900s, is shown in parentheses.

Students	Hard	Variables	Soft_Max	Soft_Min	Time_Max	Time_Min
7	225	113	18	11	0,01 (50)	0,01(50)
14	956	640	44	30	0,01 (50)	0,01(50)
21	2416	1879	67	47	0,02 (50)	0,01 (50)
28	4953	4184	91	64	0,07 (50)	0,01 (50)
35	8911	7892	118	83	0,34 (50)	0,02 (50)
42	14632	13321	145	103	1,29 (50)	0,04 (50)
49	22421	20814	170	121	2,78 (50)	0,15 (50)
56	32692	30760	203	144	15 (50)	0,59 (50)
63	45645	43457	222	159	44 (50)	0,71 (50)
70	61734	59256	252	179	83 (47)	4,92 (50)
77	81198	78445	266	190	78 (45)	8,66 (50)
84	104574	101452	296	212	97 (35)	26 (50)
91	132063	128574	323	232	182 (22)	20 (50)
98	164001	160166	350	251	182 (19)	67 (48)
105	200797	196664	374	268	292 (18)	79 (44)
112	242692	238203	396	284	281 (14)	31 (38)
119	290160	285272	426	307	236 (6)	59 (37)
126	343404	338146	452	325	422 (4)	39 (30)

mances of the maximizing and minimizing encodings are due to the number of conflicts that WPM3 must detect to find an optimal solution. As said above, an optimal solution of the maximizing encoding falsifies much more soft clauses and WPM3 usually has to solve a larger sequence of SAT problems in this case.

WPM3 has an incomplete version that stops the solver after a prefixed time. This option could be used to obtain good quality solutions when the complete version of WPM3 used in the experiments fails to find an optimal solution. The incomplete version often computes optimal solutions but it cannot certify that the solutions are optimal as the exact WPM3 version does.

5.5 Reducing team formation to TCPC

The TCPC is a particular team formation problem that only considers the preferences of the students to create teams. However, classroom team formation can involve more sophisticated and sensible criteria based, for example, on Organisational Psychology (OP). In this section, we show how a more involved OP-based problem, the Synergistic Team Composition Model (STCM) [18], can be mapped into our framework.

The dominant OP approaches to finding good teams rely on individual competences and personality traits. In the field of education, such competences refer to different types of intelligences, which can be roughly judged by teachers in order to avoid an invasive and expensive testing process. On the other hand, personality traits are usually measured through subjective self-assessment tests.

A Post-Jungian personality test is based on the cognitive mode model developed by the pioneering psychiatrist Carl Gustav Jung [114]. It has two pairs of complementary variables that determine psychological functions: Sensing/Intuition (SN) and Thinking/Feeling (TF); and two pairs of complementary variables that determine psychological attitudes: Perception/Judgment (PJ) and Extroversion/Introversion (EI). Psychological functions and attitudes form a four-dimension vector $\boldsymbol{p} = (EI, SN, TF, PJ) \in [-1, 1]^4$ that characterizes a personality. In [192, 193], Wilde proposes balancing the teams by incorporating individuals of different gender having diverse sensing/intuition and thinking/feeling, at least one introvert person and at least one extrovert, thinking and judging person.

For competences, we consider the Multiple Intelligences Theory [87]. In this theory, each person has a competence profile given by an eight-dimension vector defined as $\boldsymbol{l} = \langle vl, lm, sv, bk, mu, ie, ia, na \rangle \in [0, 1]^8$, where each dimension represents a type of intelligence. We consider the following intelligences: *vl* is verbal-linguistic intelligence, *lm* is logical-mathematical intelligence, *sv* is spatial-visual intelligence, *bk* is bodily-kinesthetic intelligence, *mu* is musical intelligence, *ie* is interpersonal intelligence, *ia* is intrapersonal intelligence and *na* is naturalist intelligence.

The goal of our problem is to create teams for performing a given task taking into account the personality and competences of individuals. A task τ requires a set of competences ($c_i \in C_\tau$), where each competence has an associated weight $w_i \in [0, 1]$ that indicates its relevance for the task fulfillment and a desired level $l_i \in [0, 1]$. Furthermore, any task has a parameter $\lambda \in [0,1]$ that balances the importance of the proficiency $u_{prof(k)}$ and congeniality $u_{con(k)}$ of team k, and is used to calculate the suitability $s(k) = \lambda \cdot u_{prof(k)} + (1-\lambda) \cdot u_{con(k)}$ of team k. Moreover, every task has a parameter $v \in [0,1]$ that balances the importance of under-competence and over-competence for the calculation of proficiency $u_{prof(k)}$, as we explain below.

A *competent student set* for a competence c_i is defined as $\delta(c_i) = \{a \in k \mid c_i \in \{l_i^a \mid l_i^a > 0\}\}$, where the zero is a typical arbitrary threshold. We define a responsibility *assignment* as a correspondence between students and required competences such that every competence is associated at least with a student in $\delta(c_i)$. We note by Θ_τ^k the set of competence assignments η for task τ and team k. The proficiency degree $\eta_{prof}(k,\tau)$ for a team k and task τ given a responsibility assignment η is one minus the sum of penalties associated to over-competence $o(\eta)$ and under-competence $u(\eta)$ of that team performing the task. We define under-competence and over-competence as follows:

$$u(\eta) = \sum_{i \in I_\tau} w_i \cdot \frac{\sum_{a \in \delta(c_i)} \mid min(l^a(c_i) - l_i, 0) \mid}{\mid \{a \in \delta(c_i) \mid l^a(c_i) - l_i \leq 0\} \mid}$$

$$o(\eta) = \sum_{i \in I_\tau} w_i \cdot \frac{\sum_{a \in \delta(c_i)} max(l^a(c_i) - l_i, 0)}{\mid \{a \in \delta(c_i) \mid l^a(c_i) - l_i \geq 0\} \mid}$$

Proficiency is defined as $u_{prof(k)} = \max_{\eta \in \Theta_\tau^k} (1 - (v \cdot u(\eta) + (1-v) \cdot o(\eta))$. Penalties are added because over-competence causes boredom and under-competence causes frustration. We note that students who are not responsible for a given competence for the task can remain free of paying penalties for that competence. Competence assignments can have different properties. In the education case, we are interested in *inclusive* assignments, which are the ones where each team member is responsible of at least one competence for the task.

Congeniality is defined as $u_{con}(k) = u_{SNTF}(k) + u_{ETJ}(k) + u_I(k) + u_{gender}(k)$, where:

- $u_{SNTF}(k) = \sigma_{SN}(k) \cdot \sigma_{TF}(k)$, the product of standard deviation for SN and TF personality components of the members of team k.

- $u_{ETJ}(k) = max_{a \in k^{ETJ}}[max((\boldsymbol{0}, \boldsymbol{\alpha}, \boldsymbol{\alpha}, \boldsymbol{\alpha}) \cdot \boldsymbol{p}, 0), 0]$, where $\boldsymbol{\alpha \approx 0.5287/3}$ and $k^{ETJ} = \{a \in k \mid tf^a > 0, ei^a > 0, pj^a > 0\}$. It is the importance of having a strong ETJ individual.

- $u_I(k) = max_{a \in k^I}[max((\boldsymbol{0}, \boldsymbol{0}, \boldsymbol{-\beta}, \boldsymbol{0}) \cdot \boldsymbol{p}, 0), 0]$, where $\boldsymbol{\beta = 3 \cdot \alpha = 0.5287}$ and $k^I = \{a \in k \mid ei^a \leq 0\}$. This is the importance of having a strong I (introvert) individual.

- $u_{gender}(k) = \gamma \cdot sin\left((\pi \cdot w(k))/(w(k) + m(k))\right)$ where $\gamma = 0.1$, $w(k)$ stands for the number of women and $m(k)$ stands for the number of men in team k. This is the importance of having a satisfactory gender balance.

Example 5.1. Assume that we want to solve a task τ with two competences, $(c_1, l_1 = 0.8, w_1 = 0.5)$ and $(c_2, l_2 = 0.6, w_2 = 0.5)$, and an under-proficiency penalty of $v = 0.6$. In fact, the competences are the set of intelligences needed for task τ. We want to find the ***inclusive*** assignment maximizing $s(k) = \lambda \cdot u_{prof(k)} + (1-\lambda) \cdot u_{con(k)}$, where we consider $\lambda = 0.5$. To evaluate the suitability of a three-student team $k = \{S_1, S_2, S_3\}$, we have:

- $\langle \boldsymbol{S_1}, woman, \langle p(sn) = 0.4,\ p(tf) = -0.4,\ p(ei) = 0.5,\ p(pj) = -0.7\rangle, [l(c_1) = 0.9, l(c_2) = 0.5]\rangle$
- $\langle \boldsymbol{S_2}, man, \langle p(sn) = -0.7,\ p(tf) = 0.6,\ p(ei) = 0.8,\ p(pj) = 0.4\rangle, [l(c_1) = 0.2, l(c_2) = 0.8]\rangle$
- $\langle \boldsymbol{S_3}, man, \langle p(sn) = 0.8,\ p(tf) = -0.7,\ p(ei) = -0.4,\ p(pj) = -0.6\rangle, [l(c_1) = 0.4, l(c_2) = 0.6]\rangle$

We want to assign students to task competences so that (1) each student is responsible for at least one competence (*inclusive*), (2) each competence is covered by at least one student (*assignment*), and (3) The proficiency degree $\eta_{prof}(k, \tau)$ for a team k and task τ given the assignment η is maximal in Θ^k_τ. For this example, we consider individual competences like an atomic task.

Table 5.3 shows every valid student assignment for both competences as well as its under-proficiency and over-proficiency penalty sum with $v = 0.6$. An assignment for both competences is not inclusive if some student has no competence assigned. The maximization of $u_{prof(k)} = \max_{\eta \in \Theta^k_\tau} (1 - (v \cdot u(\eta) + (1 - v) \cdot o(\eta))$ involves the minimization of the under-proficiency and over-proficiency penalty sum among the assignments. Table 5.4 shows every valid assignment $\eta(k, \tau)$ and, for the inclusive ones, the $cost(\eta(k, \tau)) = \sum_i cost(\eta(k, c_i))$, or total cost for the assignment.

The former problem involving minimization of costs among assignments can be efficiently solved using the minimum cost flow model [10]. The minimum cost flow problem has a time complexity of $O(m \cdot log(n) \cdot (m + n \cdot log(n)))$ on a network with n nodes and m arcs [174], where $n = |k| + |I|$ (team size and competences number in task τ) and $m = \sum_i |\delta(c_i)|$. Furthermore, this problem of cost minimization among assignments can be avoided if we consider as valid just the assignments where every student is responsible for all the task competences.

We calculate now *congeniality* $u_{con}(k) = u_{SNTF}(k) + u_{ETJ}(k) + u_I(k) + u_{gender}(k)$, where:

- $u_{SNTF}(k) = \sigma_{SN}(k) \cdot \sigma_{TF}(k) \approx 0.7767 \cdot 0.6807 \approx \mathbf{0.5287}$.
- $u_{ETJ}(k) = max_{a \in k^{ETJ}}[max((\mathbf{0}, \boldsymbol{\alpha}, \boldsymbol{\alpha}, \boldsymbol{\alpha}) \cdot \boldsymbol{p}, 0), 0]$, where $\boldsymbol{\alpha} \approx \mathbf{0.1762}$ and $k^{ETJ} = \{S_2\}$. Calculating we get $u_{ETJ}(k) = \mathbf{0.3172}$.
- $u_I(k) = max_{a \in k^I}[max((\mathbf{0}, \mathbf{0}, -\boldsymbol{\beta}, \mathbf{0}) \cdot \boldsymbol{p}, 0), 0]$, where $\boldsymbol{\beta} = \mathbf{0.5287}$ and $k^I = \{S_3\}$. Thus, $u_I(k) = \mathbf{0.2115}$.

i	$\eta(k,c_i)$ S_1	S_2	S_3	$u(\eta(k,c_i))$	$o(\eta(k,c_i))$	$cost(\eta(k,c_i))$
1	**0**	**0**	**1**	**0,12**	**0**	**0,12**
1	**0**	**1**	**0**	**0,18**	**0**	**0,18**
1	0	1	1	0,15	0	0,15
1	**1**	**0**	**0**	**0**	**0,02**	**0,02**
1	1	0	1	0,12	0,02	0,14
1	1	1	0	0,18	0,02	0,2
1	1	1	1	0,15	0,02	0,17
2	**0**	**0**	**1**	**0**	**0**	**0**
2	**0**	**1**	**0**	**0**	**0,04**	**0,04**
2	0	1	1	0	0,04	0,04
2	**1**	**0**	**0**	**0,03**	**0**	**0,03**
2	1	0	1	0,03	0	0,03
2	1	1	0	0,03	0,04	0,07
2	1	1	1	0,03	0,04	0,07

Table 5.3: For each competence c_i and student assignment $\eta(k, c_i)$, under/over-proficiency costs are calculated and added. Bold lines are base cases where only one student is responsible; the rest of lines are calculated from these base lines. Only valid assignments are shown, excluding the case $S_1 = S_2 = S_3 = 0$.

	$\boldsymbol{\eta(k, c_1)}$						
$\boldsymbol{\eta(k, c_2)}$	1	2	3	4	5	6	7
1	~~Inc~~	~~Inc~~	~~Inc~~	~~Inc~~	~~Inc~~	0.2	0.17
2	~~Inc~~	~~Inc~~	~~Inc~~	~~Inc~~	0.18	~~Inc~~	0.21
3	~~Inc~~	~~Inc~~	~~Inc~~	**0.06**	0.18	0.24	0.21
4	~~Inc~~	~~Inc~~	0.18	~~Inc~~	~~Inc~~	~~Inc~~	0.2
5	~~Inc~~	0.21	0.18	~~Inc~~	~~Inc~~	0.23	0.2
6	0.19	~~Inc~~	0.22	~~Inc~~	0.21	~~Inc~~	0.24
7	0.19	0.25	0.22	0.09	0.21	0.27	0.24

Table 5.4: Valid assignments $\eta(k, \tau)$ from $\eta(k, c_1)$ and $\eta(k, c_2)$ showing costs $u(\eta) + o(\eta)$. ~~Inc~~ stands for not inclusive. $\boldsymbol{u_{prof}(k, \tau) = 0.94}$ $(1 - 0.06)$.

- $u_{gender}(k) = \gamma \cdot sin\left((\pi \cdot w(k))/(w(k)+m(k))\right)$ where $\gamma = 0.1$, $w(k)$ stands for the number of women and $m(k)$ for the number of men in team k. Thus, $u_{gender}(k) = 0.1 \cdot sin\left(\frac{\pi}{3}\right) \approx \mathbf{0.0183}$.

We calculate $u_{con}(k) \approx 0.5287 + 0.3172 + 0.2115 + 0.0183 \approx 1.0757$ and $s(k) \approx (0.5 \cdot 0.94) + (0.5 \cdot 1.0757) \approx 1,00785$. We now multiply this number by 1000 and round to the nearest integer as it is needed by the majority of optimizers. Then we will use a final desirability degree $\boldsymbol{s(k) = 1008}$. In summary, we can use the same encoding of TCPC, adjusting the size of teams and number of members and replacing the weights in soft clauses in such a way that every possible team has as weight its desirability degree. If we had teams of different sizes, we should scale $s(k)$ as we did in the TCPC case.

5.6 Related work

There are similarities between the TCPC and the classical matching theory problem known as the Stable Roommate Problem (SRP). SRP is about finding a stable matching for an even-sized set. A matching is a partition of the set into disjoint pairs of roommates. We say a matching is stable if there are not two individuals who are not roommates and both prefer each other to their roommate under the current matching. TCPC and SRP try to match elements within a single set in groups of a given size (size two for SRP).

Irving [109] described an algorithm to solve SRP with a time complexity of $O(n^2)$. Nevertheless, this algorithm solves a decision problem. It determines whether a stable matching exists, and if so, it returns that matching. We are not solving a decision problem but an optimization one. Furthermore, the TCPC is about matching elements in groups of any given size or a combination of sizes. Standard SRP uses rooms of size two but this problem becomes NP-complete for rooms of size three [110]. The NP-completeness proof uses the partition into triangles problem as in the NP-completeness proof of TCPC.

Note that the TCPC is about finding an optimal assignment given a certain criterion, but matching theory algorithms deal with the notion of stability. An optimization version of SRP has not always an optimal solution among the stable solutions, so that they can miss optimality. We show below an example.

Given a totally ordered preference list of possible mates for each student, the desirability of A to be with B in a team k is calculated as the size for k minus the position in that list, starting by an index equal to one. Thus, the last position in the list has a desirability of zero. We show this information in the graph of Figure 5.1 for four students.

Table 5.5 shows desirability degrees for every possible partition of size two of the graph. Partition suitability is calculated as the sum of arities for each node into the subgraphs for each partition. We observe, for a standard SRP and a usual team suitability degree calculation, by summing satisfactions, that neither stability implies optimality nor optimality implies stability.

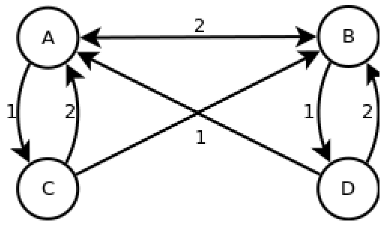


Figure 5.1: Degree of individual convenience of students in a team.

Partition	Desirability Calculation	Optimal	Stable
DA , BC	$(1+0)+(1+1)=\mathbf{3}$	✗	✗
DB , AC	$(2+0)+(1+2)=\mathbf{5}$	✓	✗
DC , AB	$(0+0)+(2+2)=\mathbf{4}$	✗	✓

Table 5.5: Optimality and stability of size-two partitions of graph in Figure 5.1.

There are criteria for the evaluation of team suitability that are different from the ones described in the previous section for STCM. Another psychological theory, proposed by Belbin [52], insists on the importance of roles in team composition processes [44]. Belbin exposes nine important roles that an individual can play in a team: plant, resource investigator, coordinator, shaper, monitor evaluator, implementer, teamworker, specialist and completer-finisher. According to Belbin's theory, people play such roles with three different performances: preferred team roles, manageable roles and least preferred roles. An effective team needs to be role balanced, having at least one individual playing any role with a given minimum performance, which can be preferred or manageable. Based exclusively on this theory, Alberola et al. [12] also pose this problem as a multi-agent coalition structure generation problem, developing a tool for practical use in an educational environment [11]. This tool uses Bayesian learning to estimate the predominant roles for each student from the peer-evaluation history made by their former teammates.

5.7 Concluding remarks

We have presented two different ways of solving the TCPC as a Weighted Partial MaxSAT problem, proved that the TCPC is NP-hard, and conducted experiments to evaluate the proposed approach using an exact MaxSAT solver. The experimental results show that the minimizing encoding outperforms the maximizing encoding, and MaxSAT solving is a good solving method for the TCPC.

The proposed MaxSAT approach has the following advantages: it is generic, and so does not need a dedicated algorithm; it is declarative, and so is easier to understand the way the problem is specified; it is flexible, and so different classroom configurations and other team formation problems can be solved similarly; and it is efficient, and so provides an optimal solution in a reasonable amount of time. Moreover, the idea of creating a minimizing MaxSAT encoding from a maximizing MaxSAT encoding is new, and can be applied to many similar optimization problems.

Chapter 6

Conclusions and Future Work

This chapter summarizes the main contributions of the thesis and outlines a few research directions that we plan to pursue in the future.

6.1 Conclusions

The scientific contributions of this PhD thesis advance the state of the art and open new research directions in MaxSAT and MinSAT solving. The thesis proposes novel logic-based methods for solving challenging optimization problems. Moreover, it reinforces the idea that MaxSAT and MinSAT are competitive generic problem solving approaches in a wide range of application domains.

The main original contributions of this PhD thesis can be summarized as follows:

- The definition of a complete tableau calculus for Weighted Partial MaxSAT.
- The definition of a complete tableau calculus for Weighted Partial MinSAT.
- The definition of a complete calculus that is valid for both Weighted Partial MaxSAT and Weighted Partial MinSAT.
- The definition of the direct, improved and Tseitin-based transformations from non-clausal MaxSAT to clausal MaxSAT, and its conversion into cost-preserving transformations from non-clausal MinSAT to clausal MinSAT.
- The definition of a complete tableau calculus for solving non-clausal MaxSAT, which can also be used to solve non-clausal MinSAT.
- The resolution of the team composition problem in a classroom as a MaxSAT problem.

We believe that the inference systems defined in the thesis have contributed to the creation of a robust logical framework for MaxSAT and MinSAT. Before starting this thesis, to the best of our knowledge, there were no tableau-based approaches for

MaxSAT and MinSAT and there was no approach for solving non-clausal MaxSAT and non-clausal MinSAT.

We also believe that the proposed tableau calculi for MaxSAT and MinSAT could be used in logic courses to illustrate the differences between decision and optimization problems from a logic perspective. Semantic tableaux are very intuitive and could be helpful to explain and understand basic concepts of model and proof theory in Boolean optimization.

It seems clear that new logical approaches to MaxSAT and MinSAT will help to improve MaxSAT and MinSAT solvers. The challenge is to find out if these logical approaches can be used to devise novel solving methods for SAT and develop more powerful SAT solvers.

Finally, we would like to highlight that the application of MaxSAT solving to the area of team formation is promising. The usual constraints in team formation admit efficient MaxSAT encodings, and the challenge is to select the right encoding for each constraint. Moreover, the insights derived from the maximizing and minimizing encodings can be used to encode other optimization problems.

6.2 Future work

There are several interesting research directions that are worth exploring in the future, including the following ones:

- The definition of MaxSAT and MinSAT tableau calculi for first-order logic is a natural extension of our work and opens a new research direction.

- The extension of MaxSAT and MinSAT tableau calculi to non-classical logics would allow to solve optimization problems in richer formalisms and deal with notions such as uncertainty and imprecision.

- The development of a robust and highly efficient MinSAT solver is crucial to develop new MinSAT applications and test some of the ideas presented in the thesis. It is also essential to better understand the duality between MaxSAT and MinSAT from a practical perspective.

- The empirical comparison of the cost-preserving transformations defined in this PhD thesis is important to identify situations in which a transformation is preferred over the others.

- The creation of a test bed of non-clausal MaxSAT and MinSAT instances is decisive to advance the state of the art on non-clausal MaxSAT and MinSAT.

- The definition of cost-preserving transformations from signed formulas to signed CNF formulas is needed to study non-clausal MaxSAT and MinSAT in many-valued logics.

- A deeper understanding of the differences between the solving techniques and encodings of MaxSAT and MinSAT is significant to develop new techniques and applications.
- The resolution of the TCPC problem with MinSAT solving and its experimental comparison with the MaxSAT approach could help to improve the scalability of the approach proposed for the TCPC.
- The definition of MaxSAT and MinSAT encodings of more complex team composition problems is key to consolidate MaxSAT and MinSAT as a competitive alternative in the area of team formation.

Bibliography

[1] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A parametric approach for smaller and better encodings of cardinality constraints. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming, CP, Uppsala, Sweden*, pages 80–96, 2013.

[2] André Abramé and Djamal Habet. Ahmaxsat: Description and evaluation of a branch and bound Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:89–128, 2014.

[3] André Abramé and Djamal Habet. Efficient application of Max-SAT resolution on inconsistent subsets. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming, CP, Lyon, France*, pages 92–107, 2014.

[4] André Abramé and Djamal Habet. Local Max-Resolution in branch and bound solvers for Max-SAT. In *Proceedings of the 26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI, Limassol, Cyprus*, pages 336–343, 2014.

[5] André Abramé and Djamal Habet. Maintaining and handling all unit propagation reasons in exact Max-SAT solvers. In *Proceedings of the 7th Annual Symposium on Combinatorial Search, SOCS, Prague, Czech Republic*, 2014.

[6] André Abramé and Djamal Habet. Local search algorithm for the partial minimum satisfiability problem. In *Proceedings of the 27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI, Vietri sul Mare, Italy*, pages 821–827, 2015.

[7] André Abramé and Djamal Habet. On the resiliency of unit propagation to Max-Resolution. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI, Buenos Aires, Argentina*, pages 268–274, 2015.

[8] André Abramé and Djamal Habet. Learning nobetter clauses in Max-SAT branch and bound solvers. In *Proceedings of the 28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI, San Jose, CA, USA*, pages 452–459, 2016.

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

[10] R. Ahuja, T. Magnanti, and J. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[11] Juan M. Alberola, Elena del Val, Víctor Sánchez-Anguix, Alberto Palomares, and Maria Dolores Teruel. An artificial intelligence tool for heterogeneous team formation in the classroom. *Knowledge-Based Systems*, 101:1–14, 2016.

[12] Juan M. Alberola, Elena del Val Noguera, Víctor Sánchez-Anguix, and Vicente Julián. Simulating a collective intelligence approach to student team formation. In *Proceedings of the 8th International Conference on Hybrid Artificial Intelligent Systems, HAIS, Salamanca, Spain*, pages 161–170, 2013.

[13] Teresa Alsinet, Felip Manyà, and Jordi Planes. Improved branch and bound algorithms for Max-SAT. In *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing, SAT, Santa Margherita Ligure, Italy*, 2003.

[14] Teresa Alsinet, Felip Manyà, and Jordi Planes. A Max-SAT solver with lazy data structures. In *Proceedings of the 9th Ibero-American Conference on Artificial Intelligence, IBERAMIA, Puebla, México*, pages 334–342, 2004.

[15] Teresa Alsinet, Felip Manyà, and Jordi Planes. Improved exact solver for Weighted Max-SAT. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing, SAT, St. Andrews, Scotland*, pages 371–377, 2005.

[16] Teresa Alsinet, Felip Manyà, and Jordi Planes. An efficient solver for Weighted Max-SAT. *Journal of Global Optimization*, 41:61–73, 2008.

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

[18] Ewa Andrejczuk, Juan A. Rodríguez-Aguilar, Carme Roig, and Carles Sierra. Synergistic team composition. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS, São Paulo, Brazil*, pages 1463–1465, 2017.

[19] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In *Proceedings of the Technical Communications of the 28th International Conference on Logic Programming, ICLP, Budapest, Hungary*, pages 211–221, 2012.

[20] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy. Improving WPM2 for (Weighted) Partial MaxSAT. In *Proceedings of the 19th International Conference om Principles and Practice of Constraint Programming, CP, Uppsala, Sweden*, pages 117–132, 2013.

[21] Carlos Ansótegui, María 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, SAT, Swansea, UK*, pages 427–440, 2009.

[22] Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial MaxSAT. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI, Atlanta, Georgia, USA*, 2010.

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

[24] Carlos Ansótegui, María Luisa Bonet, Jordi Levy, and Felip Manyà. Inference rules for high-order consistency in Weighted CSP. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence, AAAI, Vancouver, Canada*, pages 167–172, 2007.

[25] Carlos Ansótegui, María Luisa Bonet, Jordi Levy, and Felip Manyà. The logic behind Weighted CSP. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI, Hyderabad, India*, pages 32–37, 2007.

[26] Carlos Ansótegui, Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution procedures for multiple-valued optimization. *Information Sciences*, 227:43–59, 2013.

[27] Carlos Ansótegui, Frédéric Didier, and Joel Gabàs. Exploiting the structure of unsatisfiable cores in MaxSAT. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI, Buenos Aires, Argentina*, pages 283–289, 2015.

[28] 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, CPAIOR, Yorktown Heights, NY, USA*, pages 403–409, 2013.

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

[30] Carlos Ansótegui, Idelfonso Izquierdo, Felip Manyà, and José Torres Jiménez. A Max-SAT-based approach to constructing optimal covering arrays. In *Proceedings of the 16th International Conference of the Catalan Association for Artificial Intelligence, CCIA, Vic, Spain*, pages 51–59, 2013.

[31] Carlos Ansótegui, Chu Min Li, Felip Manyà, and Zhu Zhu. A SAT-based approach to MinSAT. In *Proceedings of the 15th International Conference of the Catalan Association for Artificial Intelligence, CCIA, Alacant, Spain*, pages 185–189, 2012.

[32] Josep Argelich, Alba Cabiscol, Inês Lynce, and Felip Manyà. Efficient encodings from CSP into SAT, and from MaxCSP into MaxSAT. *Multiple-Valued Logic and Soft Computing*, 19(1-3):3–23, 2012.

[33] Josep Argelich, Chu Min Li, and Felip Manyà. An improved exact solver for Partial Max-SAT. In *Proceedings of the International Conference on Nonconvex Programming: Local and Global Approaches, NCP, Rouen, France*, pages 230–231, 2007.

[34] Josep Argelich, Chu Min Li, and Felip Manyà. Exploiting many-valued variables in MaxSAT. In *Proceedings of the 47th IEEE International Symposium on Multiple-Valued Logic, ISMVL, Novi Sad, Serbia*, pages 155–160, 2017.

[35] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. The first and second Max-SAT Evaluations. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:251–278, 2008.

[36] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. Analyzing the instances of the MaxSAT Evaluation. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing, SAT, Ann Arbor, MI, USA*, pages 360–361, 2011.

[37] Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. Experimenting with the instances of the MaxSAT Evaluation. In *Proceedings of the 14th International Conference of the Catalan Association for Artificial Intelligence, CCIA, Lleida, Spain*, pages 31–40, 2011.

[38] Josep Argelich, Chu Min Li, Felip Manyà, and Joan Ramon Soler. Clause branching in MaxSAT and MinSAT. In *Proceedings of the 21st International Conference of the Catalan Association for Artificial Intelligence, CCIA, Roses, Spain*, pages 17–26, 2018.

[39] Josep Argelich, Chu Min Li, Felip Manyà, and Joan Ramon Soler. Clause tableaux for maximum and minimum satisfiability. *Logic Journal of the IGPL*, 29(1):7–27, 2021.

[40] Josep Argelich, Chu Min Li, Felip Manyà, and Zhu Zhu. MinSAT versus MaxSAT for optimization problems. In *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming, CP, Uppsala, Sweden*, pages 133–142, 2013.

[41] Josep Argelich, Chu Min Li, Felip Manyà, and Zhu Zhu. Many-valued MinSAT solving. In *Proceedings of the 44th International Symposium on Multiple-Valued Logics, ISMVL, Bremen, Germany*, pages 32–37, 2014.

[42] Josep Argelich and Felip Manyà. Exact Max-SAT solvers for over-constrained problems. *Journal of Heuristics*, 12(4–5):375–392, 2006.

[43] Josep Argelich and Felip Manyà. Partial Max-SAT solvers with clause learning. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing, SAT, Lisbon, Portugal*, pages 28–40, 2007.

[44] A. Aritzeta, S. Swailes, and B. Senior. Belbin's team role model: Development, validity and applications for team building. *Journal of Management Studies*, 44(1):96–118, 2007.

[45] Florent Avellaneda. A short description of the solver EvalMaxSAT. In *Proceedings of the MaxSAT Evaluation*, pages 8–9, 2020.

[46] 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, CP, Melbourne, VIC, Australia*, pages 641–651, 2017.

[47] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. MaxSAT Evaluation 2018: New developments and detailed results. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):99–131, 2019.

[48] 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, CP, Kinsale, Ireland*, pages 108–122, 2003.

[49] Nikhil Bansal and Venkatesh Raman. Upper bounds for MaxSat: Further improved. In *Proceedings of the 10th International Symposium on Algorithms and Computation, ISAAC, Chennai, India*, pages 247–260, 1999.

[50] Bernhard Beckert, Reiner Hähnle, and Felip Manyà. The SAT problem of signed CNF formulas. In *Labelled Deduction*, volume 17 of *Applied Logic Series*, pages 59–80. Springer, Dordrecht, 2000.

[51] R. Béjar and F. Manyà. Solving the round robin problem using propositional logic. In *Proceedings of the 17th National Conference on Artificial Intelligence, AAAI, Austin, Texas, USA*, pages 262–266, 2000.

[52] R.Meredith Belbin. *Team Roles at Work: A Strategy for Human Resource Management*. Butterworth-Heinemann, Oxford, 1993.

[53] Jeremias Berg, Fahiem Bacchus, and Alex Poole. Abstract cores in implicit hitting set MaxSat solving. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing, SAT, Alghero, Italy*, pages 277–294, 2020.

[54] Daniel Le Berre. SAT4J, a satisfiability library for Java. `http://www.sat4j.org/`, 11 2020.

[55] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–6, 2010.

[56] A. Biere and W. Kunz. SAT and ATPG: Boolean engines for formal hardware verification. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design, ICCAD, San Jose, California, USA*, pages 782–785, 2002.

[57] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58:117–148, 2003.

[58] Miquel Bofill, Dídac Busquets, Víctor Muñoz, and Mateu Villaret. Reformulation based MaxSAT robustness. *Constraints*, 18(2):202–235, 2013.

[59] Miquel Bofill, Jordi Coll, Josep Suy, and Mateu Villaret. SAT encodings of pseudo-Boolean constraints with at-most-one relations. In *Proceedings of the 16th International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR, Thessaloniki, Greece*, pages 112–128, 2019.

[60] Miquel Bofill, Marc Garcia, Josep Suy, and Mateu Villaret. MaxSAT-based scheduling of B2B meetings. In *Proceedings of the 12th International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR, Barcelona, Spain*, pages 65–73, 2015.

[61] Maria Luisa Bonet, Sam Buss, Alexey Ignatiev, João Marques-Silva, and António Morgado. MaxSAT resolution with the dual rail encoding. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, AAAI, New Orleans, Louisiana, USA*, pages 6565–6572, 2018.

[62] Maria Luisa Bonet and Jordi Levy. Equivalence between systems stronger than resolution. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing, SAT, Alghero, Italy*, pages 166–181, 2020.

[63] María Luisa Bonet, Jordi Levy, and Felip Manyà. A complete calculus for Max-SAT. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, SAT, Seattle, USA*, pages 240–251, 2006.

[64] María Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8–9):240–251, 2007.

[65] Brian Borchers and Judith Furman. A two-phase exact algorithm for MAX-SAT and Weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2:299–306, 1999.

[66] Joan Casas-Roma, Antonia Huertas, and Felip Manyà. Solving MaxSAT with natural deduction. In *Proceedings of the 20th International Conference of the Catalan Association for Artificial Intelligence, CCIA, Deltebre, Spain*, pages 186–195, 2017.

[67] Milan Češka, Jiří Matyáš, Vojtech Mrazek, and Tomáš Vojnar. Satisfiability solving meets evolutionary optimisation in designing approximate circuits. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing, SAT, Alghero, Italy*, pages 481–491, 2020.

[68] Ting Chen, Vladimir Filkov, and Steven S. Skiena. Identifying gene regulatory networks from experimental data. *Parallel Computing*, 27(1):141 – 162, 2001.

[69] Edmund Clarke, Daniel Kroening, and Flavio Lerda. A tool for checking ANSI-C programs. In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS, Barcelona, Spain*, pages 168–176, 2004.

[70] Edmund M. Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

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

[72] IBM CPLEX MIP solver. `https://www.ibm.com/es-es/products/ilog-cplex-optimization-studio`, 11 2020.

[73] Marcello D'Agostino. Tableaux methods for classical propositional logic. In *Handbook of Tableau Methods*, pages 45–123. Kluwer, 1999.

[74] Dominique D'Almeida and Éric Grégoire. Model-based diagnosis with default information implemented through MAX-SAT technology. In *Proceedings of the IEEE 13th International Conference on Information Reuse & Integration, IRI, Las Vegas, NV, USA*, pages 33–36, 2012.

[75] Sylvain Darras, Gilles Dequen, Laure Devendeville, and Chu Min Li. On inconsistent clause-subsets for Max-SAT solving. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming, CP, Providence, RI, USA*, pages 225–240, 2007.

[76] Jessica Davies. *Solving MAXSAT by Decoupling Optimization and Satisfaction*. PhD thesis, Department of Computer Science, University of Toronto, 2013.

[77] 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, CP, Perugia, Italy*, page 225–239, 2011.

[78] 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, SAT, Helsinki, Finland*, pages 166–181, 2013.

[79] 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, CP, Uppsala, Sweden*, pages 247–262, 2013.

[80] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[81] Luís Guerra e Silva, Luís Miguel Silveira, and João P. Marques Silva. Algorithms for solving Boolean satisfiability in combinational circuits. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE, Munich, Germany*, pages 526–530, 1999.

[82] Rüdiger Ehlers, Kai Treutler, and Volker Wesling. SAT solving with fragmented Hamiltonian path constraints for wire arc additive manufacturing. In *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing, SAT, Alghero, Italy*, pages 492–500, 2020.

[83] Tobias Eibach, Enrico Pilz, and Gunnar Völkel. Attacking Bivium using SAT solvers. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing, SAT, Guangzhou, China*, pages 63–76, 2008.

[84] MaxSAT Evaluation 2020: Solver and benchmark descriptions. `https://helda.helsinki.fi/bitstream/handle/10138/318451/mse20proc.pdf?sequence=1`, 11 2020.

[85] Melvin C. Fitting. *First-Order Logic and Automated Theorem Proving*. Springer, New York, second edition, 1996.

[86] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, SAT, Seattle, USA*, pages 252–265, 2006.

[87] H. Gardner and T. Hatch. Educational implications of the theory of multiple intelligences. *Educational Researcher*, 18(8):4–10, 1989.

[88] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.

[89] Carla P. Gomes, Bart Selman, Ken McAloon, and Carol Tretkoff. Randomization in backtrack search: Exploiting heavy-tailed profiles for solving hard scheduling problems. In *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems, AIPS, Pittsburg, PA, USA*, pages 208–213, 1998.

[90] 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, CP, Québec City, QC, Canada*, pages 941–956, 2012.

[91] Gurobi mip solver. `https://www.gurobi.com/`, 11 2020.

[92] Reiner Hähnle. Tableaux and related methods. In *Handbook of Automated Reasoning*, pages 100–178. Elsevier and MIT Press, 2001.

[93] Federico Heras and Javier Larrosa. New inference rules for efficient Max-SAT solving. In *Proceedings of the National Conference on Artificial Intelligence, AAAI, Boston/MA, USA*, pages 68–73, 2006.

[94] Federico Heras, Javier Larrosa, and Albert Oliveras. MiniMaxSAT: An efficient Weighted Max-SAT solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

[95] Federico Heras, António Morgado, and João Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence, AAAI, San Francisco, California, USA*, pages 36–41, 2011.

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

[97] Federico Heras, Antonio Morgado, Jordi Planes, and Joao Marques-Silva. Iterative SAT solving for minimum satisfiability. In *Proceedings of the IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI, Athens, Greece*, pages 922–927, 2012.

[98] Marijn J. H. Heule. Schur number five. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, AAAI, New Orleans, Louisiana, USA*, pages 6598–6606, 2018.

[99] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving very hard problems: Cube-and-Conquer, a hybrid SAT solving method. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI, Melbourne, Australia*, pages 4864–4868, 2017.

[100] Randy Hickey and Fahiem Bacchus. Speeding up assumption-based SAT. In *Proceedings of the 22nd International Conference on Theory and Applications of Satisfiability Testing, SAT, Lisbon, Portugal*, pages 164–182, 2019.

[101] J. N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15:359–383, 1995.

[102] Alexey Ignatiev, Antonio Morgado, Vasco Manquinho, I Lynce, and J Marques-Silva. Progression in maximum satisfiability. *Frontiers in Artificial Intelligence and Applications*, 263:453–458, 08 2014.

[103] Alexey Ignatiev, António Morgado, and João Marques-Silva. On reducing maximum independent set to minimum satisfiability. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing, SAT, Vienna, Austria*, pages 103–120, 2014.

[104] Alexey Ignatiev, António Morgado, and João Marques-Silva. On tackling the limits of resolution in SAT solving. In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing, SAT, Melbourne, Australia*, pages 164–183, 2017.

[105] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. PySAT: A Python toolkit for prototyping with SAT oracles. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing, SAT, Oxford, UK*, pages 428–437, 2018.

[106] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. RC2: an efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11:53–64, 09 2019.

[107] Alexey Ignatiev, António Morgado, Jordi Planes, and João Marques-Silva. Maximal falsifiability - definitions, algorithms, and applications. In *Proceedings of the 9th International Conference on Logic Programming and Automated Theorem Proving, LPAR, Stellenbosch, South Africa*, pages 439–456, 2013.

[108] Alexey Ignatiev, António Morgado, Jordi Planes, and João Marques-Silva. Maximal falsifiability. *AI Communications*, 29(2):351–370, 2016.

[109] Robert W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of Algorithms*, 6(4):577–595, 1985.

[110] Kazuo Iwama, Shuichi Miyazaki, and Kazuya Okamoto. Stable roommates problem with triple rooms. In *Proceedings of the 10th Korea-Japan Joint Workshop on Algorithms and Computation, WAAC, Gwangju, Korea*, pages 105–112, 2007.

[111] Saïd Jabbour, Nizar Mhadhbi, Badran Raddaoui, and Lakhdar Sais. A SAT-based framework for overlapping community detection in networks. In *Proceedings of the 21st Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, Part II, PAKDD, Jeju, South Korea*, pages 786–798, 2017.

[112] Hua Jiang, Chu Min Li, and Felip Manyà. An exact algorithm for the maximum weight clique problem of large graphs. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence, AAAI, San Francisco/CA, USA*, 2017.

[113] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

[114] C. Jung. Psychological types. Princeton University Press. *Princeton*, 1971.

[115] D. Kaiss, M. Skaba, Z. Hanna, and Z. Khasidashvili. Industrial strength SAT-based alignability algorithm for hardware equivalence verification. In *Proceedings of the 7th International Conference on Formal Methods in Computer-Aided Design, FMCAD, Austin, Texas, USA*, pages 20–26, 2007.

[116] Michał Karpiński and Marek Piotrów. Encoding cardinality constraints using multiway merge selection networks. *Constraints*, 24(3-4):234–251, 04 2019.

[117] Henry Kautz. Deconstructing planning as satisfiability. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI, Boston, Massachusetts*, pages 1524–1526, 2006.

[118] Henry A. Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI, Portland, OR, USA*, pages 1194–1201, 1996.

[119] Janez Konc and Dušanka Janežič. An improved branch and bound algorithm for the maximum clique problem. *MATCH - Communications in Mathematical and in Computer Chemistry*, 58:569–590, 01 2007.

[120] Boris Konev and Alexei Lisitsa. Computer-aided proof of Erdős discrepancy properties. *Artificial Intelligence*, 224:103–118, 2015.

[121] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A Partial Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 8(1/2):95–100, 2012.

[122] Adrian Kuegel. Improved exact solver for the Weighted MAX-SAT problem. In *Proceedings of the 1st Workshop on Pragmatics of SAT , POS, Edinburgh, UK*, pages 15–27, 2010.

[123] Adrian Kügel. Natural Max-SAT encoding of Min-SAT. In *Proceedings of the 6th International Conference on Learning and Intelligent Optimization, Revised Selected Papers, LION, Paris, France*, pages 431–436, 2012.

[124] Javier Larrosa and Federico Heras. Resolution in Max-SAT and its relation to local consistency in weighted CSPs. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI, Edinburgh, Scotland*, pages 193–198, 2005.

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

[126] Zhendong Lei and Shaowei Cai. Solving (Weighted) Partial MaxSAT by dynamic local search for SAT. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI, Stockholm, Sweden*, pages 1346–1352, 2018.

[127] Leonid A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):115–116, 1973.

[128] Chu-Min Li, Zhiwen Fang, and Ke Xu. Combining MaxSAT reasoning and incremental upper bound for the maximum clique problem. In *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI, Herndon, VA, USA*, pages 939–946, 2013.

[129] Chu-Min Li, Hua Jiang, and Felip Manyà. On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem. *Computers & OR*, 84:1–15, 2017.

[130] Chu-Min Li, Hua Jiang, and Ruchu Xu. Incremental MaxSAT reasoning to reduce branches in a branch-and-bound algorithm for MaxClique. In *Proceedings of the 9th International Conference on Learning and Intelligent Optimization, LION, Lille, France*, pages 268–274, 2015.

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

[132] Chu Min Li and Felip Manyà. An exact inference scheme for MinSAT. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence, IJCAI, Buenos Aires, Argentina*, pages 1959–1965, 2015.

[133] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Transforming inconsistent subformulas in MaxSAT lower bound computation. In *Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming, CP, Sydney, Australia*, pages 582–587, 2008.

[134] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Exploiting cycle structures in Max-SAT. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing, SAT, Swansea, UK*, pages 467–480, 2009.

[135] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in MaxSAT. *Constraints*, 15(4):456–484, 2010.

[136] Chu Min Li, Felip Manyà, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP, Sitges, Spain*, pages 403–414, 2005.

[137] Chu Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI, Boston, MA, USA*, pages 86–91, 2006.

[138] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.

[139] Chu Min Li, Felip Manyà, Zhe Quan, and Zhu Zhu. Exact MinSAT solving. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing, SAT, Edinburgh, UK*, pages 363–368, 2010.

[140] Chu Min Li, Felip Manyà, and Joan Ramon Soler. A clause tableau calculus for MaxSAT. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence, IJCAI, New York City, NY, USA*, pages 766–772, 2016.

[141] Chu Min Li, Felip Manyà, and Joan Ramon Soler. A clause tableau calculus for MinSAT. In *Proceedings of the 19th International Conference of the Catalan Association for Artificial Intelligence, CCIA, Barcelona, Spain*, pages 88–97, 2016.

[142] Chu Min Li, Felip Manyà, and Joan Ramon Soler. Clausal form transformation in MaxSAT. In *Proceedings of the 49th IEEE International Symposium on Multiple-Valued Logic, ISMVL, Fredericton, Canada*, pages 132–137, 2019.

[143] Chu Min Li, Felip Manyà, and Joan Ramon Soler. A tableau calculus for non-clausal maximum satisfiability. In *Proceedings of the 28th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX, London, UK*, pages 58–73, 2019.

[144] Chu Min Li, Felip Manyà, and Amanda Vidal. Tableaux for maximum satisfiability in Łukasiewicz logic. In *Proceedings of the 50th International Symposium on Multiple-Valued Logics, ISMVL, Miyazaki, Japan*, pages 243–248, 2020.

[145] Chu Min Li and Zhe Quan. Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In *Proceedings of the 22th IEEE International Conference on Tools with Artificial Intelligence, ICTAI, Arras, France*, pages 344–351, 2010.

[146] 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, AAAI, Atlanta, Georgia, USA*, pages 128–133, 2010.

[147] Chu Min Li, Zhu Zhu, Felip Manyà, and Laurent Simon. Minimum satisfiability and its applications. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI, Barcelona, Spain*, pages 605–610, 2011.

[148] Chu Min Li, Zhu Zhu, Felip Manyà, and Laurent Simon. Optimizing with minimum satisfiability. *Artificial Intelligence*, 190:32–44, 2012.

[149] Han Lin and Kaile Su. Exploiting inference rules to compute lower bounds for MAX-SAT solving. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI, Hyderabad, India*, pages 2334–2339, 2007.

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

[151] Inês Lynce and João Marques-Silva. Efficient haplotype inference with Boolean satisfiability. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAAI, Boston, Massachusetts*, page 104–109, 2006.

[152] Inês Lynce and João Marques-Silva. SAT in bioinformatics: Making the case with haplotype inference. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing, SAT, Seattle, WA, USA*, page 136–141, 2006.

[153] Vasco M. Manquinho, Joao Marques-Silva, and Jordi Planes. Algorithms for weighted Boolean optimization. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing, SAT, Swansea, UK*, pages 495–508, 2009.

[154] Vasco M. Manquinho, Ruben Martins, and Inês Lynce. Improving unsatisfiability-based algorithms for Boolean optimization. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing, SAT, Edinburgh, UK*, pages 181–193, 2010.

[155] Norbert Manthey. SMAX - implementing a robust MaxSAT interface. In *Proceedings of the MaxSAT Evaluation*, pages 30–31, 2020.

[156] F. Manyà, R. Béjar, and G. Escalada-Imaz. The satisfiability problem in regular CNF-formulas. *Soft Computing: A Fusion of Foundations, Methodologies and Applications*, 2(3):116–123, 1998.

[157] Felip Manyà. The 2-SAT problem in signed CNF formulas. *Multiple-Valued Logic. An International Journal*, 5(4):307–325, 2000.

[158] Felip Manyà, Santiago Negrete, Carme Roig, and Joan Ramon Soler. A MaxSAT-based approach to the team composition problem in a classroom. In *AAMAS 2017 Workshops, Visionary Papers, São Paulo, Brazil, Revised Selected Papers*, pages 164–173, 2017.

[159] Felip Manyà, Santiago Negrete, Carme Roig, and Joan Ramon Soler. Solving the team composition problem in a classroom. *Fundamenta Informaticae*, 174:83–101, 05 2020.

[160] João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms & applications. *Annals of Mathematics and Artificial Intelligence*, 62(3-4):317–343, 2011.

[161] Joao Marques-Silva and Vasco M. Manquinho. Towards more effective unsatisfiability-based maximum satisfiability algorithms. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing, SAT, Guangzhou, China*, pages 225–230, 2008.

[162] Joao Marques-Silva and Jordi Planes. On using unsatisfiability for solving maximum satisfiability. *ArXiv*, abs/0712.1097, 2007.

[163] Joao Marques-Silva and Jordi Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE, Munich, Germany*, pages 408–413, 2008.

[164] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Reflections on "incremental cardinality constraints for MaxSAT". *CoRR*, abs/1910.04643, 2019.

[165] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Exploiting cardinality encodings in parallel maximum satisfiability. In *Proceedings of the IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI, Boca Raton, FL, USA*, pages 313–320, 2011.

[166] 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, SAT, Vienna, Austria*, pages 438–445, 2014.

[167] Andrew Mihal and Steve Teig. A constraint satisfaction approach for programmable logic detailed placement. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing, SAT, Helsinki, Finland*, page 208–223, 2013.

[168] Antonio Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In *Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming, CP, Lyon, France*, pages 564–573, 2014.

[169] 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.

[170] Antonio Morgado, Alexey Ignatiev, and Joao Marques-Silva. MSCG: Robust core-guided MaxSAT solving: System description. *Journal on Satisfiability, Boolean Modeling and Computation*, 9:129–134, 12 2015.

[171] Alexander Nadel. Anytime Weighted Maxsat with improved polarity selection and bit-vector optimization. In *Proceedings of the 19th International Conference on Formal Methods in Computer Aided Design, FMCAD, San Jose, CA, USA*, pages 193–202, 2019.

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

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

[174] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Operations Research*, 41(2):338–350, 1993.

[175] Christos H. Papadimitriou. *Computational complexity*. Academic Internet Publishers, 2007.

[176] Tobias Paxian and Bernd Becker. Pacose: An iterative SAT-based MaxSAT solver. In *Proceedings of the MaxSAT Evaluation*, page 12, 2020.

[177] Marek Piotrów. UWrMaxSat: an efficient solver in MaxSAT Evaluation 2020. In *Proceedings of the MaxSAT Evaluation*, pages 34–35, 2020.

[178] Knot Pipatsrisawat, Akop Palyan, Mark Chavira, Arthur Choi, and Adnan Darwiche. Solving Weighted Max-SAT problems in a reduced search space: A performance analysis. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:191–217, 2008.

[179] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[180] Miquel Ramírez and Héctor Geffner. Structural relaxations by variable renaming and their compilation for solving MinCostSAT. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming, CP, Providence, RI, USA*, pages 605–619, 2007.

[181] Jussi Rintanen. Engineering efficient planners with SAT. In *Proceedings of the 20th European Conference on Artificial Intelligence, ECAI, Montpellier, France*, page 684–689, 2012.

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

[183] 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, SAT, Bordeaux, France*, pages 539–546, 2016.

[184] Haiou Shen and Hantao Zhang. Study of lower bound functions for MAX-2-SAT. In *Proceedings of the 19th National Conference on Artificial Intelligence, AAAI, San Jose, CA, USA*, pages 185–190, 2004.

[185] 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, CP, Sitges, Spain*, pages 827–831, 2005.

[186] Raymond Smullyan. *First-Order Logic*. Dover Publications, New York, second corrected edition, 1995. First published 1968 by Springer-Verlag.

[187] E. Tomita and T. Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In *Proceedings of the 4th International Conference on Discrete Mathematics and Theoretical Computer Science, DMTCS, Dijon, France*, pages 278–289, 2003.

[188] G.S. Tseitin. *Studies in Constructive Mathematics and Mathematical Logic, Part II*, chapter On the Complexity of Derivations in the Propositional Calculus, pages 115–125. Steklov Mathematical Institute, 1968.

[189] M.N. Velev and R.E. Bryant. Effective use of Boolean satisfiability procedures in the formal verification of superscalar and VLIW microprocessors. In *Proceedings of the 38th annual Design Automation Conference, DAC, Las Vegas, NV, USA*, pages 226–231, 2001.

[190] Y. Vizel, G. Weissenbacher, and S. Malik. Boolean satisfiability solvers and their applications in model checking. *Proceedings of the IEEE*, 103(11):2021–2035, 2015.

[191] Richard J. Wallace and Eugene Freuder. Comparative studies of constraint satisfaction and Davis-Putnam algorithms for maximum satisfiability problems. In *Cliques, Coloring and Satisfiability*, volume 26, pages 587–615. American Mathematical Society, 1996.

[192] D. Wilde. Teamology: The construction and organization of effective teams. *Springer-Verlag, London*, 2009.

[193] D. Wilde. Post-Jungian personality theory for individuals and teams. *SYDROSE LP*, 2013.

[194] Zhao Xing and Weixiong Zhang. Efficient strategies for (weighted) maximum satisfiability. In *Proceedings of the 10th International Conference on Principles*

and Practice of Constraint Programming, CP, Toronto, Canada, pages 690–705, 2004.

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

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

[197] Akihisa Yamada, Takashi Kitamura, Cyrille Artho, Eun-Hye Choi, Yutaka Oiwa, and Armin Biere. Optimization of combinatorial testing by incremental SAT solving. In *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation, ICST, Graz, Austria*, pages 1–10, 2015.

[198] Aolong Zha. QMaxSAT in MaxSAT Evaluation 2018. In *Proceedings of the MaxSAT Evaluation*, page 16, 2020.

[199] Hantao Zhang and Jieh Hsiang. Solving open quasigroup problems by propositional reasoning. In *Proceedings of the International Computer Symposium, ICS*, 1994.

[200] Hantao Zhang, Dapeng Li, and Haiou Shen. A SAT based scheduler for tournament schedules. In *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing, SAT, Vancouver, BC, Canada*, pages 10–13, 2004.

[201] Hantao Zhang, Haiou Shen, and Felip Manyà. Exact algorithms for MAX-SAT. *Electronic Notes in Theoretical Computer Science*, 86(1):190–203, 2003.

[202] Lei Zhang and Fahiem Bacchus. MAXSAT heuristics for cost optimal planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence, AAAI, Toronto, Ontario, Canada*, pages 1846–1852, 2012.

[203] Zhu Zhu, Chu Min Li, Felip Manyà, and Josep Argelich. A new encoding from MinSAT into MaxSAT. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming, CP, Québec City, QC, Canada*, pages 455–463, 2012.