Reasoning and Inference for (Maximum) Satisfiability: New Insights

Mohamed Sami Cherif

Aix*Marseille université
Socialement engagée

NNT/NL : 2022AIXM0589/033ED184

THÈSE DE DOCTORAT

Soutenue à Aix-Marseille Université
le 13 décembre 2022 par

Mohamed Sami Cherif

Reasoning and Inference for (Maximum) Satisfiability: New Insights

Discipline
Informatique

École doctorale
ED 184 Mathèmatiques et Informatique

Laboratoire/Partenaires de recherche
Laboratoire d'Informatique et Systèmes

LABORATOIRE D'INFORMATIQUE & SYSTÈMES
UMR 7020

Composition du jury

Carlos Ansótegui Universitat de Lleida, Spain	Rapporteur
Gilles Audemard Artois University, France	Rapporteur
Felip Manyà Artificial Intelligence Research Institute, CSIC, Spain	Examinateur
Christine Solnon INSA Lyon, France	Présidente
Djamal Habet Aix-Marseille University, France	Directeur de thèse
Richard Ostrowski Aix-Marseille University, France	Co-directeur de thèse

Affidavit

I, the undersigned, Mohamed Sami Cherif, hereby declare that the work presented in this manuscript is my own work, carried out under the scientific direction of Djamal Habet and Richard Ostrowski, in accordance with the principles of honesty, integrity and responsibility inherent to the research mission. The research work and the writing of this manuscript have been carried out in compliance with both the french national charter for Research Integrity and the Aix-Marseille University charter on the fight against plagiarism. This work has not been submitted previously in the same or in a similar version to any other examination body.

Marseille, 2022 October 7

List of Publications

International Journals

1. Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Proofs and Certificates for Max-SAT”. in: *J. Artif. Intell. Res.* 75 (2022), pp. 1373–1400. DOI: 10.1613/jair.1.13811

2. Mohamed Sami Cherif, Djamal Habet, and André Abramé. “Understanding the Power of Max-SAT Resolution through UP-Resilience”. In: *Artif. Intell.* 289 (2020), p. 103397. DOI: 10.1016/j.artint.2020.103397

International Conferences

1. Mohamed Sami Cherif, Djamal Habet, and Matthieu Py. “From Crossing-Free Resolution to Max-SAT Resolution”. In: *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*. Ed. by Christine Solnon. Vol. 235. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 12:1–12:17. DOI: 10.4230/LIPIcs.CP.2022.12

2. Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “Combining VSIDS and CHB Using Restarts in SAT”. in: *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*. Ed. by Laurent D. Michel. Vol. 210. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 20:1–20:19. DOI: 10.4230/LIPIcs.CP.2021.20

3. Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Inferring Clauses and Formulas in Max-SAT”. in: *33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021*. IEEE, 2021, pp. 632–639. DOI: 10.1109/ICTAI52525.2021.00101

4. Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Computing Max-SAT Refutations using SAT Oracles”. In: *33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021*. IEEE, 2021, pp. 404–411. DOI: 10.1109/ICTAI52525.2021.00066

5. Mohamed Sami Cherif, Djamal Habet, and André Abramé. *Understanding the power of Max-SAT resolution through UP-resilience*. Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21). Poster. Aug. 2021. URL: https://hal-amu.archives-ouvertes.fr/hal-03334479

6. Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “A Proof Builder for Max-SAT”. in: *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*. Ed. by Chu-Min Li and Felip Manyà. Vol. 12831. Lecture Notes in Computer Science. Springer, 2021, pp. 488–498. DOI: 10.1007/978-3-030-80223-3_33

7. Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Towards Bridging the Gap Between SAT and Max-SAT Refutations”. In: *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*. IEEE, 2020, pp. 137–144. DOI: 10.1109/ICTAI50040.2020.00032

8. Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “On the Refinement of Conflict History Search Through Multi-Armed Bandit”. In: *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*. IEEE, 2020, pp. 264–271. DOI: 10.1109/ICTAI50040.2020.00050

National Conferences

1. Mohamed Sami Cherif, Djamal Habet, and Matthieu Py. “De la résolution à la max-résolution”. In: *Journées Francophones de Programmation par Contraintes (JFPC)*. Saint-Étienne, France, June 2022

2. Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Max-réfutations et oracles SAT”. in: *Journées Francophones de Programmation par Contraintes (JFPC)*. Saint-Étienne, France, June 2022

3. Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Explication de clauses et de formules dans Max-SAT”. in: *Journées Francophones de Programmation par Contraintes (JFPC)*. Saint-Étienne, France, June 2022

4. Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Certificats d’optimalité pour Max-SAT”. in: *Journées Francophones de Programmation par Contraintes (JFPC)*. Saint-Étienne, France, June 2022

5. Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “Un bandit manchot pour combiner CHB et VSIDS”. in: *Journées Francophones de Programmation par Contraintes (JFPC)*. Online, France, June 2021

6. Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “Raffiner l’heuristique CHS à l’aide de bandits”. In: *Journées Francophones de Programmation par Contraintes (JFPC)*. Online, France, June 2021

7. Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Des réfutations SAT aux réfutations Max-SAT”. in: *Journées Francophones de Programmation par Contraintes (JFPC)*. Online, France, June 2021

Abstract

At the heart of computer science and artificial intelligence, logic is often used as a powerful language to model and solve complex problems that arise in academia and in real-world applications. A well-known formalism in this context is the Satisfiability (SAT) problem which simply checks whether a given propositional formula in the form of a set of constraints, called clauses, can be satisfied. A natural optimization extension of this problem is Maximum Satisfiability (Max-SAT) which consists in determining the maximum number of clausal constraints that can be satisfied within the formula. In our work, we are interested in studying the power and limits of inference and resoning in the context of (Maximum) Satisfiability. Our first contributions revolve around investigating inference in SAT and Max-SAT solving. First, we study statistical inference within a Multi-Armed Bandit (MAB) framework for online selection of branching heuristics in SAT and we show that it can further enhance the efficiency of modern clause-learning solvers. Moreover, we provide further insights on the power of inference in Branch and Bound algorithms for Max-SAT solving through the property of UP-resilience. Our contributions also extend to SAT and Max-SAT proof theory. We particularly attempt to theoretically bridge the gap between SAT and Max-SAT inference.

Keywords: Artificial Intelligence, SAT, Max-SAT, Inference, Reasoning

Résumé

Au cœur de l'informatique et de l'intelligence artificielle, la logique est souvent utilisée comme un langage pour modéliser et résoudre des problèmes complexes issus du milieu académique ou d'applications industrielles. Un formalisme bien connu dans ce contexte est le problème de Satisfiabilité (SAT) qui vérifie simplement si une formule propositionnelle donnée sous la forme d'un ensemble de contraintes, appelées clauses, peut être satisfaite. Une extension naturelle de SAT en problème d'optimisation est la Satisfiabilité Maximum (Max-SAT), qui consiste à déterminer le nombre maximal de contraintes clausales pouvant être satisfaites dans la formule. Dans nos travaux, on s'intéresse à l'étude du pouvoir et des limites de l'inférence et du raisonnement dans le contexte de ces deux paradigmes. Nos premières contributions tournent autour de l'étude de l'inférence dans le cadre des algorithmes de résolution pour SAT et Max-SAT. Tout d'abord, nous étudions l'inférence statistique dans le cadre des solveurs modernes pour SAT qui sont basés sur l'apprentissage de clauses. On introduit un formalisme bandit manchot pour la sélection adaptative d'heuristiques de branchement et on montre qu'un tel mécanisme permet d'améliorer l'efficacité des solveurs modernes. De plus, nous investiguons minutieusement la puissance de l'inférence dans le cadre des algorithmes de type séparation et évaluation pour Max-SAT grâce à la propriété de l'UP-résilience. Nos contributions s'étendent également à la théorie des preuves pour SAT et Max-SAT, l'un de nos objectifs majeurs étant de combler le fossé théorique entre l'inférence SAT et Max-SAT.

Mots clés : Intelligence Artificielle, SAT, Max-SAT, Inférence, Raisonnement

Acknowledgement

First and foremost, I wish to thank all the members of the COALA team of the LIS laboratory who have kindly welcomed and supported me for more than 5 years. My experience in the COALA team started with a short internship during the first year of my master degree. I never imagined at the time that, 5 years later, I would be completing my PhD in the same team. This pleasant journey would not have been possible without the help and guidance of Prof. Djamal Habet, my PhD supervisor and head of the COALA team and of the computer science and interactions department of Aix-Marseille university. Prof. Habet have been my supervisor not only during this thesis but also during two previous internships in the COALA team. My achievements would not have been possible without his precious assistance, his dedicated involvement in every step of the way throughout these last years and his constant support, nurturing and understanding. More than a supervisor, Prof. Habet has been a life mentor who provided me with advice when I most needed it and always encouraged me to strive for the best.

I would also like to thank my PhD co-supervisor, Assoc. Prof. Richard Ostrowski, and my monitoring committee members, Prof. Philippe Jégou and Prof. Chu-Min Li, for their availability and helpful assistance. My sincere thanks also go to Prof. Cyril Terrioux and Dr. Matthieu Py for their invaluable contribution and for all the meaningful discussions that we had. I would also like to express my deepest appreciation to Prof. Carlos Ansótegui, Prof. Gilles Audemard, Prof. Felip Manyà and Prof. Christine Solnon who have kindly accepted to be members of my defense committee.

In addition, I would like to extend my sincere thanks to all the teaching staff of Aix-Marseille university and especially all the professors that taught me during my master degree. In particular, the funding of this thesis would not have been possible without the decisive role played by Prof. Nadia Creignou, head of the computer science and mathematics doctoral school, and Yvon Berland, former president of Aix-Marseille university. My teaching experience within Aix-Marseille university these last three years have also been fulfilling and I am very grateful for all the help I got for my senior colleagues.

Finally, I cannot begin to express my thanks to my family, especially my father, my mother and my sister, for always believing in me and for the strength they give me each day to thrive and strive forward. My heartfelt thanks also go to Insaf Tanazefti, Ridha Kerkeni, Soumaya Lahbib and all my friends who supported me and made every single day of my life livelier and more meaningful.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ACC	Almost Common Clauses
ACIDS	Average Conflict-Index Decision
AI	Artificial Intelligence
ASP	Answer Set Programming
BESS	Bandit Ensemble for parallel SAT Solving
BnB	Branch and Bound
CC	Configuration Checking
CCA	Configuration Checking with Aspiration
CDCL	Conflict Driven Clause Learning
CHB	Conflict History Branching
CHS	Conflict-History Search
CNF	Conjunctive Normal Form
CP	Constraint Programming
CSP	Constraint Satisfaction Problem
CVIG	Clause-Variable Incidence Graph
DAG	Directed Acyclic Graph
DCCA	Double Configuration Checking with Aspiration
DLCS	Dynamic Largest Combined Sum
DLIS	Dynamic Largest Individual Sum
DLM	Discrete Lagrangian Method
DLS	Dynamic Local Search
DP	Davis-Putnam algorithm
DPLL	Davis-Putnam-Logemann-Loveland algorithm
DUC	Dominant Unit Clause
EVSIDS	Exponential Variable State Independent Decaying Sum
ExC	Explanation Calculus
EXP	Exponential class

EXP3	Exponential-Weight Algorithm for Exploration and Exploitation
FL	Failed Literals
FM	Fu and Malik algorithm
FPT	Fixed-Parameter Tractable
FUIP	First Unit Implication Point
GUP	Generalized Unit Propagation
HT	Head/Tail
HUP	Hard Unit Propagation
IHS	Implicit Hitting Set
ILP	Integer Linear Programming
IS	Inconsistent Subset
JW	Jeroslow-Wang
LB	Lower Bound
LBD	Literal Block Distance
LRB	Learning Rate Branching
LSU	Linear SAT-UNSAT algorithm
MAB	Multi-Armed Bandit
Max-Clique	Maximum Clique
Max-CSP	Maximal Constraint Satisfaction
Max-SAT	Maximum Satisfiability
Min-SAT	Minimum Satisfiabiliy
moms	Maximum Occurrences on Minimum Sized clauses
MOSS	Minimax Optimal Strategy in the Stochastic Case
MPRO	Multiple Path Resolvent Order
NP	Non-deterministic Polynomial class
NVSIDS	Normalized Variable State Independent Decaying Sum
P	Polynomial class
PAWS	Pure Additive Weighting Scheme
PL	Pure Literals

PRCO	Path Resolvent Circular Order
PRO	Path Resolvent Order
QCC	Quantitative Configuration Checking
QSAT	Quantified Satisfiability
RD	Random strategy
RL	Reinforcement Learning
RPO	Reverse Propagation Order
RR	Round-Robin strategy
SAPS	Scaling And Probabilistic Smoothing
SAT	Satisfiability
SDF	Smoothed Descent and Flood
SIR	Smallest Intermediary Resolvent
SLS	Stochastic Local Search
SMT	SAT Modulo Theories
SS	Single Switch strategy
SUP	Simulated Unit Propagation
TS	Thompson Sampling
TSP	Traveling Salesman Problem
UB	Upper Bound
UCB	Upper Confidence Bound
UCS	Unit Clause Subset
UIP	Unit Implication Point
UP	Unit Propagation
VBS	Virtual Best Solver
VIG	Variable Incidence Graph
VMTF	Variable Move-To-Front
VSIDS	Variable State Independent Decaying Sum
WL	Watched Literals
WLCC	Watched List with Conflicts' Counter

Introduction

Since ancient times, humans have always sought to understand the laws of thought and to formalize reasoning on factual knowledge. The emergence of computer science as a full-fledged field helped to deepen our understanding of automating reasoning through effective computation. One of the main challenges in this field is to simulate the learning and inference process that the human brain can naturally and intuitively perform. This led to the advent of Artificial Intelligence (AI) as a modern discipline aiming to further investigate how machines can simulate human intelligence. In his work on the psychology of judgment and decision-making [Kah11], Kahneman describes two different modes of thought, one which is governed with rational, controlled, and factual reasoning while another relies on an automatic, quite instinctive approach to judgment and decision making. These systems intuitively correspond to the two main branches of AI. The first branch, i.e., symbolic AI, relies on formal and logical reasoning through the manipulation of symbols used to represent knowledge while the second branch, i.e., numeric AI, relies on statistical and probabilistic techniques to infer information from plainly represented data. Our work in this dissertation mainly falls within the first branch of AI relying on symbolism but it also partly relies on techniques and formalisms that originate from numeric AI.

As human society is constantly evolving, it is often confronted with different problems that arise in academia and in real-world applications. To give an example, checking whether there exists a valid bus route that visits each specific location once within a city is a typical problem that can arise in transportation. Another related problem is computing the shortest route that the bus can take among all the possible valid ones. Note how these problems can be very complex, typically when we consider a large set of locations to be visited. They are also of different nature as the former is a decision problem whose solution is either affirmative or negative while the latter is an optimization problem which requires computing the optimal route with respect to distance among all the valid ones. To solve such problems in the literature, we rely on well-established generic formalisms that can be used to model and solve problems of similar nature. One of the most common and natural ways to represent knowledge in symbolic AI is through logic as a highly expressive language. In particular, propositional logic is a simple yet powerful tool that enables to represent and deal with knowledge in the form of propositions, connected through logical operators [Boo54].

A well-known formalism in this context is the Satisfiability (SAT) problem which simply checks whether a given propositional formula in the form of a set of constraints can be satisfied [BHM21]. The formula is defined over a set of boolean variables which can only have values True or False while the constraints take the form of clauses which can contain variables or their negation. SAT is at the heart of symbolic AI and complexity theory as it is the first problem shown to be NP-complete [Coo71]. This problem also gained

particular interest for its wide range of applications and its powerful solving methods. In particular, the introduction of the Conflict Driven Clause Learning (CDCL) algorithm [MLM09], which incorporates extensive learning during the search process, led to the emergence of highly competitive modern solvers able to deal with large formula containing millions of variables and clauses. Another formalism closely related to SAT is Maximum Satisfiability (Max-SAT) [BJM21]. This problem is a natural optimization extension of SAT which consists in determining the maximum number of clausal constraints that can be satisfied in the formula. Max-SAT has many solving paradigms including Branch and Bound (BnB) algorithms [LM21], which perform an exhaustive search relying heavily on inference rules, and SAT-based algorithms [BJM21] which take advantage of the power of modern SAT solvers as decision engines. Our contributions in this manuscript revolve around these two problems. In particular, we are interested in studying the power and limits of inference and reasoning in the context of (Maximum) Satisfiability, both in theory and in practical solving.

Our first contribution falls within the scope of improving the efficiency of modern SAT solvers through incorporating stochastic reasoning formalisms originating from numeric AI and, more specifically, Reinforcement Learning (RL) [SB98]. This well-known paradigm of machine learning focuses on how an autonomous agent should learn from its prior actions and is often studied through the lens of the Multi-Armed Bandit (MAB) problem. We specifically focus on an important component in CDCL solvers which is the branching heuristic (used to pick the next variable to branch on in the search tree) and we study whether statistical inference within a MAB framework can be incorporated into such solvers to perform adaptive selection of branching heuristics. We perform an extensive evaluation of different strategies to combine the two dominant branching heuristics in the state-of-the-art, namely VSIDS [Mos+01] and CHB [Lia+16a]. We show that our MAB framework is able to achieve considerable gain by relying on Upper Confidence Bound (UCB) strategies [ACF02; AB09] enabling to conduct efficient inference based on the mean reward of each heuristic.

Our second contribution focuses on understanding the power of inference and its limits in the context of BnB algorithms for Max-SAT. Such algorithms construct a search tree and compute, at each node, a Lower Bound (LB) estimation by counting the number of disjoint Inconsistent Subsets (ISs) of the formula. When an IS is found, it is either temporarily deleted or transformed to ensure that it will be counted only once in the LB estimation [LMP07; HLO08; Li+10a; AH14d; AH14b; Abr15]. However, learning transformations of ISs by memorizing them in the current subtree may negatively affect the quality of the LB estimation. Therefore, state-of-the-art solvers learn transformations selectively mainly in the form of inference patterns. Recently, André and Habet introduced a new property, called UP-resilience, to characterize the transformations of ISs in the context of Max-SAT BnB [Abr15; AH15b]. Our work is focused on further investigating this property in order to provide a theoretical explanation for the efficiency of learning mechanisms used in BnB solvers in the last decade. In particular, we study the relation between UP-resilience and the Unit Clause Subset (UCS) patterns introduced in [AH14d; Abr15]. We assess the limits of such learning mechanisms and we provide insights to extend them through UP-resilience.

Our contributions also extend to proof theory for Max-SAT. One of the first proof systems

for Max-SAT is based on the Max-SAT resolution inference rule [BLM07; LHG08], which is an extension of the resolution rule introduced in the context of SAT [Rob65]. Max-SAT resolution is sound and complete for Max-SAT and has gained particular interest these last years [LR20a; LR20b; BL20; Fil+20]. However, we still lack understanding of many aspects related to Max-SAT resolution as an inference rule and as a proof system for Max-SAT. SAT and Max-SAT are strongly related and share many aspects. However, unlike SAT inference, Max-SAT resolution has unique features as it enforces a transfer of knowledge. Consequently, bridging the gap between SAT and Max-SAT inference remains one of the main challenges in the last decade. In this context, we contribute to the open question of whether it is possible to adapt SAT proofs into Max-SAT proofs without substantially increasing their size. We show that this is possible for certain refinements of resolution, both when Max-SAT resolution is augmented with other inference rules or when it is exclusively used in the resulting proofs. Our work in this regard led to the introduction of the first independent proof builder for Max-SAT, called MS-Builder. The proof builder relies on some of the adaptations that we introduced to compute certificates (i.e., proofs of the optimal solution) for Max-SAT instances through iterative calls to a SAT oracle. Our work also extends beyond the relative (refutational) power of proof systems for Max-SAT with their counterpart in SAT. More specifically, we introduce the notion of explainability in order to investigate the ability to deduce information within the Max-SAT paradigm and to further assess the inferential power of Max-SAT proof systems.

Our work has been published in different venues including international journals (AIJ, JAIR) [CHA20; PCH22d], international conferences (IJCAI, SAT, CP, ICTAI) [CHT20; PCH20; PCH21a; PCH21b; PCH21d; CHA21; CHT21a; CHP22b] and national conferences (JFPC) [CHT21c; CHT21d; PCH21c; PCH22a; PCH22b; PCH22c; CHP22a]. In particular, we mention that our contributions in Chapter 6 are based on joint work with Matthieu Py. Some known results are thus succinctly described in this chapter as full details can be found in [Py21]. Other publications include our solver and benchmark descriptions submitted to the SAT competitions [CHT21b; CHT21e; CHT22a; CHT22b]. Our solver Kissat_MAB won gold medals in the Main and Main SAT tracks of the 2021 SAT competition [1] as well as silver and bronze medals respectively in the Anniversary SAT and Main UNSAT tracks of the 2022 SAT competition [2].

This manuscript is organized into two parts. The first part, composed of the first three chapters, is dedicated to the necessary background on the problems we will be studying in our work. In Chapter 1, we recall preliminary definitions and notions related to computational problem solving and propositional logic. Chapters 2 and 3 include detailed overviews on SAT and Max-SAT. We provide the formal definition of both problems and we introduce their variants, their main solving methods and their main inference rules and proof systems. The second part of the manuscript is dedicated to our contributions and also includes three sections. In Chapter 4, we introduce and evaluate our MAB framework for adaptive SAT branching. We investigate the inference mechanisms employed in BnB solvers through the UP-resilience property in Chapter 5. In Chapter 6, we study the refutational and inferential

[1] results available on https://satcompetition.github.io/2021/
[2] results available on https://satcompetition.github.io/2022/

power of proof systems for Max-SAT in an attempt to deepen our knowledge on Max-SAT reasoning and its relation to SAT inference. We also finish the manuscript with a general conclusion summarizing our results and discussing future research directions.

Background

1 Preliminaries

Contents

In this Chapter, we present preliminary notions necessary for reading this manuscript. We define decision and optimization problems and we recall major notions in algorithmics, computability and complexity theory. We also introduce propositional logic and the Conjunctive Normal Form (CNF) which will be used in the problems that we will study in our work.

1.1 Computational Problems

In theoretical computer science, studying and efficiently solving academic and real-world problems requires their formalization into clearly established computational problems free of any superfluous information. An input to a computational problem is referred to as an instance of the problem. A computational problem can therefore be viewed as a set of instances corresponding each to a, possibly empty, set of solutions. Computational problems are divided into several categories, two of which we will focus on in our work namely decision and optimization problems. Note that we only consider problems of combinatorial nature, in which the set of instances is a discrete finite set of objects.

In a decision problem, the required output can be expressed in the form of a question whose answer is "yes" or "no". More formally, given a decision problem π, the set of instances of the problem $I(\pi)$ can be divided into two disjoints sets $I^+(\pi)$ and $I^-(\pi)$ representing respectively the set of positive instances for which the answer is "yes" and the set of negative instances for which the answer is "no". We provide hereafter a formal definition as well as some examples of well-known decision problems.

Definition 1.1 (Decision problem)**.** *Let π be a problem and $I(\pi)$ be the set of instances of π. We say that π is a decision problem if it has the following form:*

> ***Decision Problem*** π
> *Input:* $x \in I(\pi)$
> *Question:* $x \in I^+(\pi)$*?*

Example 1.1 (Primality Testing)**.** *A natural number n is a prime number if it has exactly two distinct divisors: 1 and n. The following problem which consists in testing whether a natural number is prime is a well-known decision problem. Note that the set of instances of this problem corresponds to the set of natural numbers, i.e., $I(\textit{Primality Testing}) = \mathbb{N}$, while the set of positive instances corresponds to the set of natural prime numbers, i.e., $I^+(\textit{Primality Testing}) = \{n \in \mathbb{N} \mid n \textit{ is prime}\}$. This problem has many applications mainly in cryptography [Mas20].*

> ***Primality Testing Problem***
> *Input:* $n \in \mathbb{N}$
> *Question: is n a prime number?*

Example 1.2 (Hamiltonian Path/Cycle)**.** *A graph G is traceable if there exists a Hamiltonian path in G, i.e., a path which visits each vertex exactly once. For instance, the graph G_1 represented on the left in Figure 1.1 is traceable since $A-B-C-D$ is a Hamiltonian path in G_1, while G_2 represented on the right does not allow such a path and therefore is not a traceable graph. The Hamiltonian path problem, defined below, is a well established decision problem which consists in checking whether a graph is traceable. Note that the set of instances of this problem corresponds to the set of graphs while the set of positive instances corresponds to the set of traceable graphs. A variant of this problem is the Hamiltonian cycle problem defined below. A Hamiltonian cycle is a path that starts and ends at the same vertex and includes every other vertex exactly once. A graph containing such a cycle is a Hamiltonian graph. These problems have several applications in vehicle route planning and biology among others [Mas16; Kim+17; MP21].*

> ***Hamiltonian Path Problem***
> *Input: a graph G*
> *Question: is G traceable?*

> ***Hamiltonian Cycle Problem***
> *Input: a graph G*
> *Question: is G Hamiltonian?*

(G_1) (G_2)

Figure 1.1: A traceable (resp. non-traceable) graph G_1 (resp. G_2).

Next, we introduce the second category of problems: optimization problems. The task in such problems consists in searching for the best evaluation of feasible solutions with respect to a given measure and goal. We provide below a formal definition as well as some known examples. Note that a feasible solution is optimal if it achieves the optimum value, i.e., the goal specified in the output, and such a solution is often provided alongside or instead of [1] the optimum.

Definition 1.2 (Optimization Problem)**.** *Let π be a problem and $I(\pi)$ be the set of instances of π. We say that $\pi = (I(\pi), f, m, g)$ is an optimization problem if it is of the following form:*

> ***Optimization Problem*** π
> *Input:* $x \in I(\pi)$
> *Output:* $opt(x) = g\{m(x,y) \mid y \in f(x)\}$

where:

- f *maps each instance* $x \in I(\pi)$ *to its set of feasible solutions* $f(x)$
- $m(x,y)$ *denotes a measure of the feasible solution* $y \in f(x)$
- g *is the goal function, either min (minimize) or max (maximize).*

Example 1.3 (Shortest Path)**.** *Given graph $G(V,E)$ where V and E respectively denote the set of vertices and the set of edges, the length of a path p from a vertex $v_1 \in V$ to a vertex $v_2 \in V$ in G, denoted $l(G,p)$, is the number of its edges. The shortest path problem defined below, which consists in finding the length of the shortest path between two vertices in a given graph, is a well established problem. Note that the set of feasible solutions for a given graph G are the available paths from v_1 to v_2 in G. The used measure is the length of the paths and the goal is clearly minimization. This problem has many real-world applications including mapping, social network analysis and logistics among others [Shu12; BSK11; Hai09].*

> ***Shortest Path Problem***
> *Input: a graph* $G = (V,E)$ *and* $(v_1, v_2) \in V^2$
> *Output:* $min\{l(G,p) \mid p$ *is a path from* v_1 *to* v_2 *in* $G\}$

Example 1.4 (Traveling Salesman)**.** *The Traveling Salesman Problem (TSP) is a well-known optimization problem which, given a list of cities and the distances between them, consists in finding the shortest possible route that visits each city exactly once and returns to the origin city. As showcased below, this problem can be modeled as a weighted graph problem which consists in finding the minimum Hamiltonian cycle [2] weight, where the weight of a path is simply the sum of the weights of its edges. This problem has many applications in scheduling*

[1] in such case, we are dealing with a search variant of the optimization problem which may also be associated with a different third category of computational problems called search problems.
[2] refer to Example 1.2

and routing among others [Pun07].

> ***Traveling Salesman Problem***
> *Input: a graph* $G = (V, E)$ *and a weighting function* $w_G : E \longrightarrow \mathbb{R}_+$
> *Output:* $min\{w_G(p) \mid p$ *is a Hamiltonian cycle in* $G\}$

1.2 Algorithmics

Algorithmics is a broad field of study within computer science which delves into the fundamental question of designing a correct method for solving a given problem. Such a method will be referred to as an "algorithm". The term is derived from the name of the 9^{th} century Persian mathematician Al-Khwārizmī [3] who introduced systematic solutions for linear and quadratic equations [KR18]. Although, historically, the design and use of systematic procedures date further back to 2500 BC, when Babylonians used rigorous procedures for performing arithmetic operations [Cha99].

Definition 1.3 (Algorithm)**.** *An algorithm is a finite sequence of well-defined instructions used to solve a specific problem.*

When designing an algorithm we are usually interested in the following properties:

- **Termination:** It consists in determining whether the algorithm halts for each input. A mathematical proof is usually provided as a termination proof, often invoking the monotonic behaviour of certain bounded measures used in the algorithm as an argument.
- **Correctness:** It consists in determining whether the algorithm is correct, i.e., generates the correct output with respect to its input and specification. If this property is associated with termination, we are dealing with total correctness. A mathematical proof may be provided to prove correctness although complex algorithms/systems may require more adapted formal and rigorous reasoning methods such as Hoare logic [Hoa69; Pie+21].
- **Efficiency:** It relates to the consumption of resources by the algorithm, mainly time and memory, with respect to the input size. This property pertains more specifically to complexity theory which will be introduced in Section 1.4.

Example 1.5. *A simple method for solving the Primality Testing Problem [4] is described in Algorithm 1.1 [5]. The algorithm is correct since all possible non-trivial divisors of the input n are tested. It also terminates since it loops on a finite set of numbers.*

[3] Muhammad IbnM ūsā Al-Khwārizmī (780 - 850)
[4] refer to Example 1.1
[5] The modulo (mod) operator returns the remainder of the Euclidean division of n by i. Therefore, checking if n mod $i = 0$ stands is equivalent to testing if i divides n.

Algorithm 1.1: Algorithm for Primality Testing

Input: a natural number n
Output: $True$ if n is prime, $False$ otherwise

```
1: for i = 2 to n − 1 do
2:     if n mod i = 0 then
3:         return False
4: end for
5: return True
```

1.3 Computability

Computabiliy is a broad subject of study which is at the heart of computer science theory. Indeed, computability is the ability to solve a problem in an effective manner which is inextricably linked to the existence of an algorithm to solve it. The study of algorithms and of computability in a broader sense as a full-fledged field of study started in the 1930s when logicians tried to understand the limits of reasoning automation. At the time, several attempts to formalise the notion of computability were made by Gödel [6] (μ-recursive functions alongside Herbrand [7] and later Kleene [8]) [Sie05; Kle37], Church [9] (λ-calculus) [Chu36] and Turing [10] (Turing machines) [Tur37b]. These systems intuitively represent universal computational models which are able to perform any task that is achievable by a computer. More specifically, the intuitive notion of an algorithm corresponds to the effective calculation performed by a Turing machine. This last statement is referred to as the Church-Turing thesis or conjecture and is strongly corroborated by the fact that the three mentioned models are equivalent [11] [Kle36; Tur37a].

Next, we give a brief overview [12] of Turing machines, introduced in 1936 [13]. Turing informally describes his model in [Tur48] as follows:

> *"[...] an infinite memory capacity obtained in the form of an infinite tape marked out into squares, on each of which a symbol could be printed. At any moment there is one symbol in the machine; it is called the scanned symbol. The machine can alter the scanned symbol, and its behavior is in part determined by that symbol, but the symbols on the tape elsewhere do not affect the behavior of the machine. However, the tape can be moved back and forth through the machine, this being one of the elementary operations of the machine."*

[6] Kurt Gödel (1906 - 1978)
[7] Jacques Herbrand (1908 - 1931)
[8] Stephen Cole Kleene (1909 - 1994)
[9] Alonzo Church (1903 - 1995)
[10] Alan Mathison Turing (1912 - 1954)
[11] which remained valid for later formal attempts to characterize computability [Aho+74; Sch80]
[12] refer to the following textbooks [LP97; BBJ07; Dav13b] for a broader overview of Turing machines and computability theory
[13] although formally published in [Tur37b] a year later

Such a machine, formally defined below, starts from the initial state q_0, scanning the leftmost input symbol. At each computation step, it evolves with respect to its current state and the scanned symbol as specified by the transition function. The machine stops when it enters a final (accepting) state, in which case the input word is accepted. Note that the word is considered rejected if the machine halts [14] on a non-final state or if it runs indefinitely (infinite loop). The set of words accepted by a Turing machine M is called the language of M, denoted $L(M)$.

Definition 1.4 (Turing Machine [Tur37b])**.** *A Turing machine is a tuple* $(Q, \Gamma, \Sigma, q_0, \delta, F)$ *where:*

- Q *is a finite set of states*
- Γ *is a finite set of alphabet symbols containing a blank symbol* $b \in \Gamma$
- $\Sigma \subseteq \Gamma \setminus \{b\}$ *is a finite set of input symbols*
- $q_0 \in Q$ *is the initial state*
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ *is the transition function where* $\leftarrow$ *(resp.* $\rightarrow$*) denotes a left (resp. right) shift*
- $F \subseteq Q$ *is the set of final (or accepting) states*

Example 1.6. *We consider the Turing machine* $M = (Q, \Gamma, \Sigma, q_0, \delta, F)$ *where:*

- $Q = \{q_0, q_1\}$ *and* $F = \{q_1\}$
- $\Sigma = \{0, 1\}$ *and* $\Gamma = \{0, 1, \sqcup\}$
- $\delta(q_0, 0) = (q_0, 0, R)$ *and* $\delta(q_0, 1) = (q_1, 1, R)$

The machine M accepts the words that contain the symbol "1" at least once, Indeed, as soon as such a symbol is encountered, M transitions to the accepting state q_1*. Formally, we have* $L(M) = \Sigma^* \setminus \{0\}^*$*. Note that the machine always halts (on finite inputs) since only right shifts are performed on the tape.*

For a decision problem, the notion of computability corresponds to the notion of Turing-decidability. More specifically, a problem is decidable if there exists a corresponding Turing machine which halts on every input, either accepting or rejecting it. There exists many known undecidable problems such as the halting problem [Tur37b; Kle52] which, given a Turing machine and an input, consists in determining whether the machine will halt or loop forever.

Note that the Turing machine described in Definition 1.4 is deterministic as its transition function associates to each (state,symbol) pair, for which it is defined, a single unique (state,symbol,shift) triplet. On the other hand, the transition function in a non-deterministic Turing machine, defined below, associates each (state,symbol) pair to a set of

[14] occurs when δ is not defined on the current (state,symbol) pair

(state,symbol,shift) triplets, enabling many choices at each computation step. Intuitively, such a machine is able to duplicate itself at each step thus creating different branches. Furthermore, an input word is accepted if one of the branches leads to an accepting state.

Definition 1.5 (Non-Deterministic Turing Machine)**.** *A non-deterministic Turing machine is a Turing machine* $M = (Q, \Gamma, \Sigma, q_0, \delta, F)$ *with a non-deterministic transition function* $\delta : Q \times \Gamma \longrightarrow \mathscr{P}(Q \times \Gamma \times \{\leftarrow, \rightarrow\})$ [15].

The execution time of a (deterministic or non-deterministic) Turing machine is measured in terms of the number of its transitions. Finally, it is important to note that the non-deterministic variant as well as many others such as multi-tape variants [Pap94] remain equivalent to deterministic Turing machines in terms of computational power [16], making Turing's model sturdy with respect to moderate changes and further corroborating the Church-Turing thesis.

1.4 Complexity Theory

Complexity Theory is a major sub-filed of theoretical computer science which focuses on studying and classifying decidable computational problems with respect to their resource usage. This involves studying and analyzing the complexity of algorithms designed to solve a specific problem in terms of resource consumption, mainly time and space required during their execution with respect to the size of the input. Note that this analysis may be performed in the best, average and worst cases. Hereafter, we will focus on the worst case time complexity [17], i.e., the highest possible execution time of an algorithm with respect to all possible inputs.

One may argue that the highly increasing efficiency of computer processors may render the analysis and the optimization of algorithm complexity unimportant or outdated. On the contrary, this unceasing gain in machine performance and technological advancement generally entails dealing with inputs of much larger size which may thus lead to a substantially higher resource consumption as showcased in Table 1.1. Therefore, when analyzing the complexity of an algorithm, we are interested in its asymptotic behavior in terms of execution time, i.e., typically when dealing with inputs of large size. More specifically, we want to bound the number of elementary instructions [18] executed by the algorithm with respect to the size of its input. The Landau [19] notations, and particularly the big O notation defined below, are often used to this end. The main asymptotic complexities are reported in Table 1.2. It is important to note that the complexity of an algorithm should be established with respect to a reasonable machine encoding of the input as showcased in Example 1.7.

Definition 1.6 (Big O)**.** *Let* $f, g : \mathbb{N} \longrightarrow \mathbb{R}_+$ *be two functions.* f *is dominated by* g*, denoted* $f = O(g)$*, if* $\exists\, c \in \mathbb{R}_+^*$ *and* $n_0 \in \mathbb{N}$ *s.t.* $\forall n \geq n_0, f(n) \leq c * g(n)$.

[15] Given a set S, $\mathscr{P}$ denotes the power set of S, i.e., the set of all subsets of S.
[16] recognize the same class of languages
[17] refer to the following textbooks [KT05; Gol08; Pap94] for a broader overview of complexity theory
[18] simple operations which are independent of the actual implementation of the algorithm
[19] Edmund Landau (1877-1938)

Input size n	Running Time					
	n	$n\,log_2 n$	n^2	n^3	2^n	$n!$
10	0	0	0	0	0	4 s
100	0	0	0	1 s	36 y	∞
1,000	0	0	1 s	18 m	10^{17} y	∞
10,000	0	0	2 m	12 d	∞	∞
100,000	0	2 s	3 h	32 y	∞	∞
1,000,000	1 s	20 s	12 d	31,710 y	∞	∞

Table 1.1: Running times rounded up in seconds (s), minutes (m), days (d) or years (y) on a processor executing a million instructions per second with respect to increasing input size and running-time bounds as reported in [KT05]. ∞ (resp. 0) denotes running times greater than 10^{25} years (resp. less than 1 second).

Notation	Name
$O(1)$	constant
$O(log\,n)$	logarithmic
$O(n)$	linear
$O(n\,log\,n)$	quasilinear
$O(n^2)$	quadratic
$O(n^3)$	cubic
$O(n^c)$	polynomial
$O(c^n)$	exponential
$O(n!)$	factorial

Table 1.2: Complexity of algorithms with respect to their asymptotic behavior

Example 1.7. *We consider Algorithm 1.1 for primality testing*[20]*. The algorithm performs at most $n-2$ iterations in the loop with a division test of constant time $c>0$ in each iteration plus a single return instruction (either in or after the loop). Therefore, the number of iterations can be bound with respect to n as follows: $C(n)=c.n+1=O(n)$. However, the complexity of this algorithm is not linear in this case. Indeed, a natural number n is usually represented in the machine by its digits and therefore its actual size is $s=O(log\,n)$* [21]*. The complexity of Algorithm 1.1 is therefore exponential as $C(s)=O(e^s)$.*

Next, we focus on the complexity of computational problems which can be very difficult to establish as it pertains to all possible algorithms that could be used to solve a problem. More specifically, establishing the exact complexity of a problem usually requires proving upper and lower complexity bounds which coincide. While the former is easier to prove as it corresponds to the worst time complexity of a specific valid algorithm solving the problem, the latter usually involves a much stronger argument through mathematical reasoning over all possible algorithms that solve the problem. As few problems have known exact

[20] refer to Examples 1.1 and 1.5

[21] More specifically, it is represented by its digits in the binary base which amount to $s=\lfloor log_2 n\rfloor+1=O(log\,n)$.

complexities [22], mainly due to the difficulty of establishing their lower bound complexity, it is more common to classify the problems with respect to their upper bound, i.e., the worst time complexity of the best know algorithm. Note that, in the rest of this section, we solely consider decision problems but complementary notions exist for optimization problems [BCC94].

We introduce below three main complexity classes: P (Polynomial), NP (Non-deterministic Polynomial) and EXP (Exponential). Intuitively, the class P contains "easy" or feasibly computable problems, which can be solved in polynomial time. This correspondence between the informal notion of "easiness" or "feasibility" with the formal complexity class P was first asserted by Cobham [Cob65] and Edmonds [Edm65] and is referred to as the Cobham–Edmonds thesis. On the other hand, the EXP class contains problems which are extremely difficult and can only be solved in exponential time. The NP class introduced by Cook [23] in [Coo71] is an intermediate class containing problems for which no polynomial algorithm is known and it further seems very difficult to prove that such algorithms do not exist. Intuitively, this class represents the problems which are polynomially verifiable, i.e., the validity of a solution certificate can be asserted in polynomial time as showcased in Example 1.9. These three classes thus form the hierarchy established in Proposition 1.1.

Definition 1.7 (P)**.** *The class P is the class of problems that can be solved in polynomial time by a deterministic Turing machine.*

Definition 1.8 (NP [Coo71])**.** *The class NP is the class of problems that can be solved in polynomial time by a non-deterministic Turing machine.*

Definition 1.9 (EXP)**.** *The class EXP is the class of problems that can be solved in exponential time by a deterministic Turing machine.*

Example 1.8. *There exists a polynomial-time algorithm solving the primality testing problem introduced in Example 1.1 and therefore it is in P [AKS04].*

Example 1.9. *There is no known polynomial algorithm for the Hamiltonian path problem introduced in Example 1.2. This problem is in NP. Indeed, if the input graph is traceable, exhibiting a Hamiltonian path is sufficient to certify its traceability. Note that the size of such a path is exactly the number of vertices in the input graph G and therefore its size is linear in the size of G. Furthermore, it is possible to check that the given path is a Hamiltonian path in polynomial time with respect to the size of G. To this end, we can simply verify that each consecutive pair of vertices in the path is linked by an edge and that all the vertices are distinct. The same arguments hold to establish that the Hamiltonian cycle problem is also in NP.*

Proposition 1.1. $P \subseteq NP \subseteq EXP$.

[22] Simple examples include finding the minimum value within a list (linear complexity) or performing a comparison sort (quasilinear complexity [Cor+09]).

[23] Stephen Cook (born in 1939)

The notion of polynomial-time reduction, introduced by Karp [24] [Kar72] and defined below, allows to transform the instances of a problem into the instances of another while preserving positive instances. Such reductions have also enabled to identify the hardest problems in NP, referred to as NP-complete problems and defined below. Many NP-complete problems are known in the literature such as those in Example 1.10. Note that the NP-completeness of a problem $\pi \in$ NP is usually established through Proposition 1.2, i.e., by proving that a known NP-complete problem can be polynomially reduced to π.

Definition 1.10 (Polynomial-Time Reduction [Kar72])**.** *Let π_1 and π_2 be two decision problems. A polynomial-time reduction from π_1 to π_2 is a function $R : I(\pi_1) \longrightarrow I(\pi_2)$ computable in polynomial-time which preserves positive instances:*

$$\forall x \in I(\pi_1), x \in I^+(\pi_1) \iff R(x) \in I^+(\pi_2)$$

Notation 1.1. *We denote $\pi_1 \leq_p \pi_2$ if there exists a polynomial time reduction from a problem π_1 to a problem π_2,*

Definition 1.11 (NP-hardness)**.** *A decision problem π is NP-hard if $\forall \pi' \in NP$, $\pi' \leq_p \pi$.*

Definition 1.12 (NP-completeness)**.** *A decision problem π is NP-complete if $\pi \in NP$ and is NP-hard.*

Example 1.10. *The Hamiltonian path and cycle problems introduced in Example 1.2 are NP-complete [GJ79].*

Proposition 1.2. *Let π_1 and π_2 be two decision problems. If π_1 is NP-difficult and $\pi_1 \leq_p \pi_2$ then π_2 is NP-difficult.*

Finally, we must mention that the relation between some complexity classes are still not clearly established. For instance, one of the major open problems in computer science is the **P vs NP** problem [25] which consists in proving or disproving P=NP. Such a result would have many implications in complexity theory as showcased in Figure 1.2 and also in fields such as cryptography, artificial intelligence and philosophy among many others.

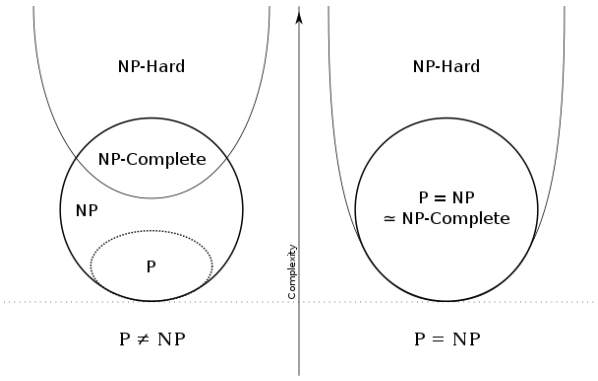


Figure 1.2: Euler diagram on the implications of P≠ NP (left) and P=NP (right) [26]

[24] Richard Manning Karp (born in 1935)

[25] one of the Millennium Prize problems selected by the Clay Mathematics Institute which are stated at https://www.claymath.org/millennium-problems

1.5 Propositional Logic

As the study of the laws of thought and of valid reasoning, logic has always been a subject of great interest which ancient and modern philosophers alike sought to investigate and formalize [KK84]. Nowadays, this field of study is at the heart of several disciplines including philosophy, mathematics and computer science. In particular, propositional logic [Boo54], which deals with knowledge in the form of propositions connected through logical operators, is of great importance in theoretical computer science and it is innately linked to the classical problems which we will study in our work.

Propositional logic is defined on the following alphabet:

- **Constants:** $\top$ (True or 1) and $\bot$ (False or 0)
- **Propositional/Boolean variables:** variables that can only have the Boolean values True or False
- **Logical connectives:** $\neg$ or $\overline{\ }$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\Rightarrow$ (implication) and $\Leftrightarrow$ (equivalence)
- **Priority symbols:** (and)

We define propositional formulas below using this alphabet. An assignment maps each variable to a truth value, i.e., a Boolean value of either True or False. In the following sections, the term assignment is used to refer to either a complete or partial assignment unless otherwise specified.

Definition 1.13 (Propositional formula)**.** *Let V be a set of propositional variables. A propositional formula on V is inductively defined as follows:*

- $\top, \bot$ *and* $x \in V$ *are propositional formulas*
- *if* ϕ *and* ψ *are propositional formulas then* (ϕ), $\neg\phi$, $\phi \wedge \psi$, $\phi \vee \psi$, $\phi \Rightarrow \psi$ *and* $\phi \Leftrightarrow \psi$ *are propositional formulas.*

Example 1.11. $\phi = \neg(x_1 \wedge x_2) \wedge (x_2 \Leftrightarrow x_3)$ *is a propositional formula over the set of variables* $V = \{x_1, x_2, x_3\}$.

Definition 1.14 (Assignment)**.** *Let V be a set of propositional variables. An assignment of variables in V is a function $\alpha : V \longrightarrow \{True, False\}$. If α assigns a Boolean value to all the variables in V, it is a complete assignment; otherwise it is a partial assignment.*

Given an assignment α of the variables in V and using the truth tables of logical connectives reported in Table 1.3, the evaluation of a propositional formula ϕ with respect to α, denoted $[\![\phi]\!]_\alpha$ or simply $\alpha(\phi)$, can be inductively deduced as follows:

[26] Figure made by Behnam Esfahbod, available under the Creative Commons Licence Attribution-ShareAlike 3.0 Unported (CC BY-SA 3.0).

ϕ	ψ	$\neg\phi$	$\phi \wedge \psi$	$\phi \vee \psi$	$\phi \Rightarrow \psi$	$\phi \Leftrightarrow \psi$
False	False	True	False	False	True	True
False	True	True	False	True	True	False
True	False	False	False	True	False	False
True	True	False	True	True	True	True

Table 1.3: Truth tables of logical connectors

- $[\![\top]\!]_\alpha = True$ and $[\![\bot]\!]_\alpha = False$
- $[\![x]\!]_\alpha = \alpha(x)$ for $x \in V$
- $[\![\neg\phi]\!]_\alpha = \neg[\![\phi]\!]_\alpha$
- $[\![\phi * \psi]\!]_\alpha = [\![\phi]\!]_\alpha * [\![\psi]\!]_\alpha$ for $* \in \{\wedge, \vee, \rightarrow, \Leftrightarrow\}$
- $[\![(\phi)]\!]_\alpha = [\![\phi]\!]_\alpha$

Example 1.12. *We consider the formula ϕ over $V = \{x_1, x_2, x_3\}$ in Example 1.11 and the assignment α which maps each variable in V to the value False (i.e., $\forall x \in V, \alpha(x) = False$). We have $[\![\phi]\!]_\alpha = \neg(\alpha(x_1) \wedge \alpha(x_2)) \wedge (\alpha(x_2) \Leftrightarrow \alpha(x_3)) = True$.*

If there exists an assignment α under which the evaluation of a propositional formula is True, we say that the formula is satisfiable and we refer to α as model of ϕ otherwise the formula is unsatisfiable. If any assignment of the variables is a model, the formula is referred to as a tautology. In addition, two formulas are logically equivalent if they are evaluated to the same truth value under any assignment of the variables.

Definition 1.15 (Model)**.** *Let ϕ be a propositional formula over the set of variables V. A model of ϕ is an assignment α s.t. $[\![\phi]\!]_\alpha = True$.*

Definition 1.16 (Tautology)**.** *A propositional formula ϕ is a tautology if any assignment of its variables is a model of ϕ.*

Definition 1.17 (Equivalence)**.** *Two propositional formulas ϕ and ψ are equivalent if $\phi \Leftrightarrow \psi$ is a tautology.*

Notation 1.2. *we denote $\phi \equiv \psi$ if ϕ and ψ are two equivalent propositional formulas.*

Example 1.13. *Let ϕ, ψ and γ be three propositional formulas. We have the following established classical equivalences:*

- ***Double negation law:*** $\neg(\neg\phi) \equiv \phi$
- ***De Morgan laws:*** $\neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi$ *and* $\neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi$
- ***Distributivity laws:*** $\phi \vee (\psi \wedge \gamma) \equiv (\phi \vee \psi) \wedge (\phi \vee \gamma)$ *and* $\phi \wedge (\psi \vee \gamma) \equiv (\phi \wedge \psi) \vee (\phi \wedge \gamma)$

- ***(bi)conditional laws:*** $\phi \Rightarrow \psi \equiv \neg\phi \vee \psi$ *and* $\phi \Leftrightarrow \psi \equiv (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$

Next, we introduce a classic subset of Boolean formulas which can be expressed through a specific, easier to handle, syntactic form. To this end, given a set of variables V, we start by defining below the basic forms that can be found in such formulas: literals and clauses.

Definition 1.18 (Literal)**.** *A literal is either a variable* $x \in V$ *or its negation* $\overline{x}$*.*

Definition 1.19 (Polarity)**.** *A literal* l *is a positive literal if it has a positive polarity, i.e.* l *is a variable. Otherwise,* l *has a negative polarity and is a negative literal.*

Definition 1.20 (Clause)**.** *A clause is a disjunction of literals.*

Notation 1.3. *Given a literal* l*, a clause* C *and a propositional formula* ϕ*, we denote* $var(l)$, $var(C)$ *and* $var(\phi)$ *respectively the variables in* l, C *and* ϕ*.*

Example 1.14. $C = \overline{x_1} \vee x_2$ *is a clause containing the negative literal* $\overline{x_1}$ *and the positive literal* x_2*. Furthermore, we have* $var(C) = \{x_1, x_2\}$*.*

Note that an assignment α of the variables in V can be represented as a set of literals. More specifically, $\alpha = \{x \in V \mid \alpha(x) = True\} \cup \{\overline{x} \mid x \in V \text{ and } \alpha(x) = False\}$. A clause $C = l_1 \wedge ... \wedge l_n$ can also be represented as set of literals $C = \{l_1, ..., l_n\}$. The evaluation of literals and clauses under assignments is very simple. Indeed, a literal l is satisfied (resp. falsified) by an assignment α if $l \in \alpha$ (resp. $\overline{l} \in \alpha$). A clause C is satisfied by an assignment α if at least one of its literals is satisfied by α, i.e., $\exists l \in C$ s.t. $l \in \alpha$; otherwise it is falsified by α. Note that some interesting relations can occur between two clauses, mainly subsumption and opposition defined below. Furthermore, specific clauses include:

- the empty clause, denoted $\square$, which contains zero literals and is always falsified
- unit (resp. binary) clauses which contain exactly one literal (resp. two literals)
- tautological clauses, which contain a literal and its negation and are always satisfied

Definition 1.21 (Subsumption)**.** *A clause* C *subsumes a clause* C' *if* $C \subseteq C'$*.*

Definition 1.22 (Opposition)**.** *A clause* C *opposes a clause* C' *if* $\exists l \in C$ *s.t.* $\overline{l} \in C'$*.*

Example 1.15. *We consider the clauses* $C_1 = x_1$, $C_2 = x_1 \vee x_2$ *and* $C_3 = \overline{x_1} \vee x_2$*. The unit clause* C_1 *clearly subsumes the binary clause* C_2 *as every literal in* C_1 *is also in* C_2*.* C_1 *also opposes the binary clause* C_3 *as* $x_1 \in C_1$ *and* $\overline{x_1} \in C_2$*.*

Finally, we introduce the Conjunctive Normal Form (CNF), in which formulas are simply conjunctions of clauses. A CNF formula ϕ can also be represented as a set of clauses and is satisfied by an assignment α if all its clauses are satisfied by α. The empty formula $\phi = \emptyset$ contains zero clauses and is always satisfied. It is important to note that every propositional formula can be transformed into an equivalent CNF formula using truth tables or the classical equivalences in Example 1.13. However, A transformation with guaranteed efficiency may warrant the addition of new variables while only preserving satisfiability instead of equivalence [Tse83; PG86].

Definition 1.23 (Conjunctive Normal Form)**.** *A propositional formula in Conjunctive Normal Form (CNF) is a conjunction of clauses.*

Notation 1.4. *Let ϕ be a CNF formula and l be a literal. We denote $\phi|_l$ the simplification of ϕ by the literal l. Formally, we have:*

$$\phi|_l = \{C \setminus \{\bar{l}\} \mid C \in \phi \text{ and } l \notin C\}$$

We can extend this notation to any assignment α as follows:

$$\phi|_\alpha = \{C \setminus \{l \in C \mid \bar{l} \in \alpha\} \mid C \in \phi \text{ and } C \cap \alpha = \emptyset\}$$

Example 1.16. *The formula $\psi = (\overline{x_1} \vee \overline{x_2}) \wedge (\overline{x_2} \vee x_3) \wedge (x_2 \vee \overline{x_3})$ is a satisfiable formula in CNF form. Indeed, the assignment α which maps each variable in V to the value False clearly satisfies each clause in ϕ and therefore is a model of ϕ. Note that this formula is equivalent to the formula ϕ in Example 1.11.*

1.6 Conclusion

In this chapter, we introduced some preliminary notions which form the necessary generic background for reading this manuscript. We started with the nature of computational problems and we particularly focused on decision and optimization problems. We also recalled some notions related to algorithmics and computability. Then, we recalled the main complexity classes and some major results in complexity theory. Finally, we introduced propositional logic alongside the Conjunctive Normal Form (CNF) which we will consider in the rest of the manuscript. In the following chapter, we introduce the Satisfiability (SAT) problem which is a well-known decision formalism at the heart of AI and complexity theory.

2 Satisfiability

Contents

In this chapter, we define the Satisfiability (SAT) problem as well as some of its known variants. We also overview the main complete and incomplete methods for SAT solving. Furthermore, we recall the major notions in proof theory for SAT. We particularly focus

on the resolution rule as a proof system for SAT and we introduce some of its well-known refinements. Finally, we define the Multi-Armed Bandit (MAB) problem and we review its use in the context of Satisfiability and beyond.

2.1 Definition and Variants

The Satisfiability (SAT) problem [1] defined below is at the heart of theoretical computer science. It is the first problem that was proven NP-complete [Coo71] and many other classical problems in the literature were later shown NP-complete by reduction from SAT [Kar72]. There is no known polynomial algorithm for the SAT problem and finding such an algorithm entails P=NP, thus solving the **P vs NP** problem. However, it is widely believed that no such algorithm exists which is equivalent to the claim P≠NP, although no valid mathematical proof has been exhibited to support this claim.

Definition 2.1 (Satisfiability)**.** *The satisfiability (SAT) problem is defined as follows:*

> ***SAT***
> *Input: a CNF formula ϕ*
> *Question: is ϕ satisfiable?*

Theorem 2.1 (Cook's Theorem [Coo71])**.** *The SAT problem is NP-complete.*

There are many known variants of the SAT problem, two of which are defined below: k-SAT and Horn-SAT. These two variants take specific formulas as input. While k-SAT restricts the size of the clauses in the input CNF formula to at most k-literals, Horn-SAT considers formulas where each clause contains at most one positive literal. Interestingly, 2-SAT and Horn-SAT are in P [Kro67; DG84]. Similar cases, referred to as traceable classes, have been disclosed in the literature [Lew78; CH91; BHS94; Sch+95].

Definition 2.2 (k-CNF Formula)**.** *Let k be a natural number. A k-CNF formula ϕ is a CNF formula s.t. each clause $C \in \phi$ contains at most k literals.*

Definition 2.3 (k-Satisfiability)**.** *Let k be a natural number. The k-satisfiability (k-SAT) problem is defined as follows:*

> ***k-SAT***
> *Input: a k-CNF formula ϕ*
> *Question: is ϕ satisfiable?*

Definition 2.4 (Horn Formula)**.** *A Horn formula ϕ is a CNF formula s.t. each clause $C \in \phi$ is Horn, i.e., contains at most one positive literal.*

Definition 2.5 (Horn-Satisfiability)**.** *Let k be a natural number. The Horn-satisfiability (Horn-SAT) problem is defined as follows:*

> ***Horn-SAT***
> *Input: a Horn CNF formula ϕ*
> *Question: is ϕ satisfiable?*

[1] also referred to as Propositional/Boolean Satisfiability problem

The SAT problem also has many known extensions such as Quantified Satisfiability (QSAT) [2], Satisfiability Modulo Theories (SMT) [3] and Sharp Satisfiability (#SAT) [4] among others. The reader can refer to the *Handbook of Satisfiability* [BHM21] for further details on these classical problems as they are out of the scope of this thesis. A particular extension we will be studying however is Maximum Satisfiability (Max-SAT), the natural optimization extension of SAT which will be introduced in Section 3.

Satisfiability is a powerful formalism that can be used to model and solve many real-world and crafted problems making it of great academic and industrial interest. It has a wide range of applications in various fields which include, but are not limited to, bounded model checking [Bie21], hardware and software verification [GGW06; Kro21], electronic design automation [MS00], planning [Rin21], data mining [Bou+18], combinatorial design theory [Zha21], cryptography [SNC09b; LJH14], statistical physics [Alt+21], and bioinformatics [LM06; MOV07]. The extensions of SAT mentioned above allow further flexibility and expressiveness thus enabling to model a larger set of problems.

A major event is held annually since 2002 [5] to keep up the driving force in improving SAT solving. SAT competitions [6] have become traditional venues to present, evaluate and compare the latest SAT solving technologies [Jär+12]. In the main track [7], solvers are evaluated on a benchmark set with varied real-word and crafted instances and are required to conform to the DIMACS format [8] and to output certificates of unsatisfiability [9].

2.2 Algorithms

In this section, we present major methods for SAT solving in the literature which can be divided into two main categories: complete and incomplete algorithms. Unlike complete methods for SAT which are guaranteed to terminate and to return a correct solution stating whether the given instance is satisfiable or unsatisfiable, incomplete methods are tailored for satisfiable instances only and are not guaranteed to find a solution. Hereafter, we overview the main complete and incomplete algorithms for SAT as well as the techniques and heuristics used in modern SAT solvers.

[2] extends SAT in the context of quantified propositional logic where each variable can be bound through existential or universal quantifiers

[3] extends SAT in the context of first-order logic with respect to a variety of theories, i.e., variables are replaced by predicates in the underlying theory which can be bound using quantifiers

[4] natural extension of SAT as a counting problem which consists in determining the number of models of the input formula

[5] 24 events have been held including three earlier SAT competitions in 1992, 1993 and 1996. 15 SAT competitions, 5 SAT races (2006, 2008, 2010, 2015, 2019) and one SAT Challenge (2012) have been held after 2000.

[6] https://satcompetition.github.io/

[7] Other previous and current tracks include the no-limits track, the parallel track, the cloud track and the random track.

[8] http://www.satcompetition.org/2009/format-benchmarks2009.html

[9] Refer to Section 2.3 for solution certification

2.2.1 Complete Methods

In this section, we present the major complete algorithms for solving the SAT problem. These algorithms explore the search space through an exhaustive branching over the variables in the formula. One particular algorithm which is at the core of every modern SAT solver is Conflict Driven Clause Learning (CDCL) [MLM09]. We start by introducing its ancestor, DPLL [DLL62], which is a simple backtracking algorithm relying mainly on the propagation of literals within unit clauses to prune the search space. Then, we present the CDCL framework which relies on two powerful mechanisms: clause learning and non-chronological backtracking. Furthermore, we overview additional techniques and heuristics which are incorporated within modern solvers to enhance their performance. Finally, we briefly introduce other complete solving methods for SAT.

2.2.1.1 DPLL Algorithm

The Davis–Putnam–Logemann–Loveland (DPLL) Algorithm [10] [DP60; DLL62] is a complete procedure for SAT solving, which paved the way for modern complete algorithms. A recursive version of this procedure is given in Algorithm 2.1. DPLL performs a systematic search by enumerating all possible assignments of the variables. To this end, a search tree where each node corresponds to a branching over a variable in the formula is constructed (lines 7-8). If all variables are assigned without falsifying a clause, then the formula is clearly satisfiable and the algorithm outputs the value $True$ (lines 3-4). In order to prune the search space, DPLL uses two simple yet powerful simplification rules (lines 5-6) which we present below.

Unit Propagation (UP) If there is a unit clause $C = l$ in the formula, it can only be satisfied by propagating the literal l, i.e., assigning l to $True$ which is equivalent to adding it to current assignment. In DPLL, this rule is enforced through simplifying the current formula by the literal l. In practice, this often leads to cascades of unit clauses being generated and to many literals being propagated, thus pruning a large part of the search space.

Pure Literals (PL) When a literal l appears with the same polarity (either positively or negatively) in the formula, then it can also be propagated. Simplifying the formula by l entails deleting the clauses containing l and therefore pruning the search space.

The above rules are performed in each node of the search tree. Furthermore, when an assignment leads to the falsification of a clause in the formula, i.e., a clause becomes empty, the current assignment cannot satisfy the formula (lines 1-2) and the algorithm backtracks to the last assigned literal in order to branch on its negation (line 8). Note that the decisions, i.e., branching on the variables, can be done simply in lexicographic or random order of the variables [DLL62]. However, more efficient branching heuristics are used in practice

[10] named after Martin Davis (born in 1928), Hilary Putnam (1926-2016), George Logemann (1938-2012) and Donald W. Loveland (born in 1934)

and will be presented in Section 2.2.1.3.2. We provide below an example to illustrate the execution of the DPLL algorithm.

Algorithm 2.1: Davis–Putnam–Logemann–Loveland (DPLL) Algorithm

Input: a CNF formula ϕ
Output: $True$ if ϕ is satisfiable, $False$ otherwise

1: **if** $\square \in \phi$ **then**
2: **return** $False$
3: **if** $\phi = \emptyset$ **then**
4: **return** $True$
5: **if** ϕ contains a unit clause or a pure literal l **then**
6: **return** $DPLL(\phi|_l)$
7: $l \longleftarrow$ Branching_Heuristic(ϕ)
8: **return** $DPLL(\phi \wedge l)$ or $DPLL(\phi \wedge \bar{l})$

Example 2.1. *We consider the formula* $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ *where* $C_1 = x_1 \vee \overline{x_2}$, $C_2 = \overline{x_2} \vee x_3$, $C_3 = \overline{x_3} \vee x_4$, $C_4 = \overline{x_3} \vee \overline{x_4}$ *and* $C_5 = \overline{x_1} \vee x_2 \vee x_3$. *A search tree depicting the execution of the DPLL algorithm on* ϕ *with lexicographic branching (i.e.,* $x_1 < x_2 < x_3 < x_4$*) is showcased in Figure 2.1.*

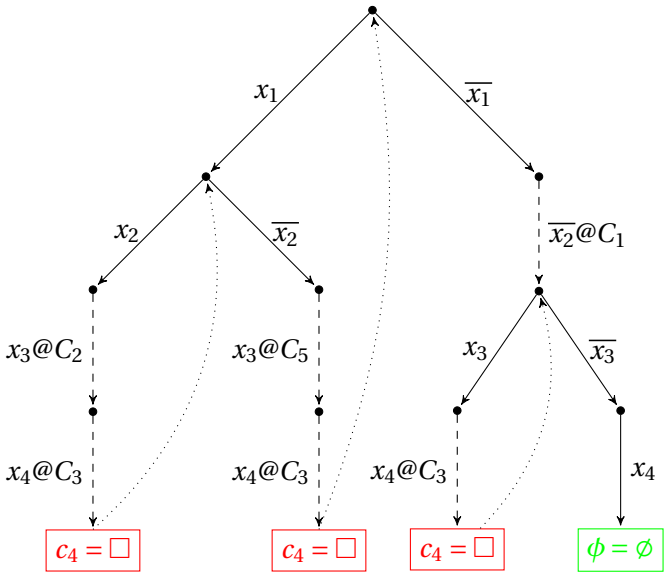


Figure 2.1: Search tree depicting the execution of the DPLL algorithm where branching is made in lexicographic order of the variables. Solid, dashed and dotted arrows represent respectively branching, propagation and backtracking. $l@C$ denotes the propagation of literal l through the clause C.

2.2.1.2 CDCL Algorithm

Modern complete SAT solver are mainly based on the Conflict Driven Clause Learning (CDCL) algorithm which is a refinement of the DPLL procedure [MLM09]. Notable CDCL solvers include GRASP [Mar95; SS96], (z)Chaff [Mos+01; Zha+01; MFM05], MiniSat [11] [ES03a], Glucose [12] [AS09a], Lingeling [13] [Bie10b], MapleSAT solver series [14] [Lia+16a; Lia+16b; Lia+16c], Cadical [15] [Bie17] and Kissat [16] [Bie+20] among many others [PD07b; Bie08b; SNC09a; Xia+19a]. The CDCL algorithm performs a similar search to DPLL by branching over the variables in the formula to enumerate all possible assignments. CDCL also relies on Unit Propagation (UP) as in DPLL to deduce information and prune the search space when possible. Furthermore, CDCL has one major additional feature compared to DPLL: clause learning.

In CDCL, we say that a conflict occurs when propagating a literal falsifies a clause in the formula. When such a case is encountered during the search, CDCL learns information in the form of a new clause, called learnt clause, to catch the cause of the failure and avoid repeating it again [Mar95; SS96]. To this end, the algorithm retraces and analyzes its actions done prior to the conflict. Formally, an implication graph, defined below, is used to represent the assignments made by the algorithm either through decisions or propagations. Decision levels of the literals are also usually reported in the implication graph. Each time a literal l is branched on, a new decision level $\delta(l) \in \{1, \ldots, |var(\phi)|\}$ is associated to l. On the other hand, the decision level of a propagated literal l corresponds to the level of the last literal assigned by branching before the propagation of l or 0 if l was propagated through a unit clause in the formula. Formally, if l is propagated through the clause C then we have $\delta(l) = \max(\{0\} \cup \{\delta(l') \mid \overline{l'} \in C \setminus \{l\}\})$.

Definition 2.6 (Implication Graph [Mar95; SS96])**.** *Let ϕ be a CNF formula and α be a partial assignment falsifying a clause $C \in \phi$. An implication graph $G = (V, A)$ is a directed labeled acyclic graph where $V = \alpha \cup \{\kappa\}$ and $A \subseteq V \times V \times \phi$ such that:*

- *If a clause $C' \in \phi$ lead to the propagation of literal $l \in C'$ then for each $l' \in C \setminus \{l\}$, we have $(\overline{l'}, l, C') \in A$. Vertices corresponding to decisions have no predecessors.*
- *The conflict is represented by the vertex $\kappa \in V$ having incoming arcs labeled by the clause C from the literals forcing it to be falsified.*

Example 2.2. *We consider the same formula $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ in Example 2.1, where $C_1 = x_1 \vee \overline{x_2}$, $C_2 = \overline{x_2} \vee x_3$, $C_3 = \overline{x_3} \vee x_4$, $C_4 = \overline{x_3} \vee \overline{x_4}$ and $C_5 = \overline{x_1} \vee x_2 \vee x_3$, and the search tree shown in Figure 2.1. The implication graph representing the assignments that lead to the first conflict is represented in Figure 2.2.*

[11] http://minisat.se/
[12] https://www.labri.fr/perso/lsimon/research/glucose/
[13] http://fmv.jku.at/lingeling/
[14] https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/
[15] http://fmv.jku.at/cadical/
[16] http://fmv.jku.at/kissat/

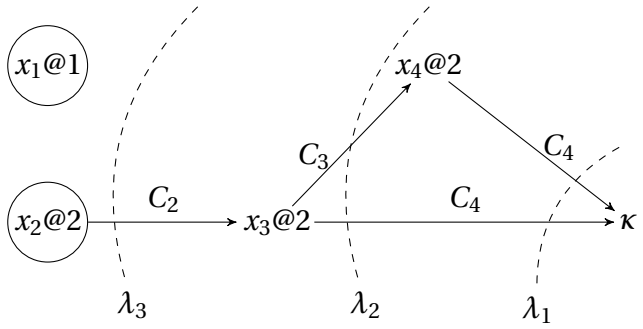


Figure 2.2: Implication graph representing assignments leading to the first conflict in Figure 2.1. Circled nodes represent literals assigned through branching. $l@\delta$ denotes the assignment of literal l at decision level δ. Dashed lines represent different cuts of the implication graph.

Using the implication graph $G = (V, A)$, we can identify the reason(s) of the conflict which take the form of a partial assignment α that generated it. More specifically, α can be obtained by identifying a bipartition (also called cut) of the implication graph such that one partition has all the decision nodes (reason side) and the other contains the conflict node κ (conflict side). The set α of vertices on the reason side that have at least one edge to the conflict side constitute a possible reason for the conflict. Since assigning the literals of α generates the same conflict, the clause $C = \{\bar{l} \mid l \in \alpha\}$ can be added to the formula to avoid reaching it again. Formally, the clause C can be deduced by the resolution rule defined below [17]. Given two opposed clauses, this well-known inference rule for SAT deduces a new clause, called resolvent, which can be added to the formula. A proof of the validity of the learnt clause can be provided by iteratively applying resolution on the clauses between the conflict node κ and the literals in α in the reverse order of propagation.

Definition 2.7 (Resolution [Rob65])**.** *Given two opposed clauses C_1 and C_2, the resolution rule is defined as follows:*

$$\frac{C_1 = x \vee A \qquad C_2 = \overline{x} \vee B}{C_3 = A \vee B}$$

Example 2.3. *We consider the implication graph in Example 2.2. The cuts λ_1, λ_2 and λ_3 correspond to possible conflict reasons $\alpha_1 = \{x_3, x_4\}$, $\alpha_2 = \{x_3\}$ and $\alpha_3 = \{x_2\}$. While α_1 represents the clause C_4, α_2 and α_3 represent respectively the clauses $C_{\alpha_2} = \overline{x_3}$ and $C_{\alpha_3} = \overline{x_2}$ which can be learned in the formula. In Figure 2.3, we showcase how these clauses can be deduced using the resolution rule.*

[17] We give a more detailed overview of resolution as a proof system for SAT in Section 2.3.

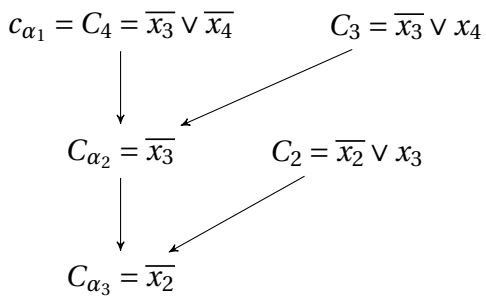


Figure 2.3: Deducing the learnt clause through resolution in CDCL

Since there can be many possible reasons to the same conflict corresponding to different cuts of the implication graph, CDCL solvers mainly use the First Unique Implication Point (FUIP) scheme to deduce and learn a new clause [Zha+01]. More specifically, this scheme consists in resolving the clauses from the conflict node until the First UIP node which is the UIP closest to κ in the implication graph. It was shown that this scheme is quite robust and effective and can provide smaller reasons for the conflicts [Zha+01].

Definition 2.8 (Unique Implication Point [Mar95; SS96])**.** *Let G be an implication graph. A Unique Implication Point (UIP) is any node in G such that any path from the decision nodes the conflict node κ must pass through it.*

Example 2.4. *The nodes x_2 and x_3 are both UIPs of the implication graph represented in Figure 2.2. Since x_3 is closest to the conflict node, it is the First UIP (FUIP) and, therefore, the clause $C_{\alpha_2} = \overline{x_3}$ will be learned in the formula.*

After determining the new clause C which is added to the formula, CDCL backtracks to the second largest decision level in C [Zha+01]. Formally, given the current decision level δ in which the conflict is detected, the backtrack level is obtained as follows:

$$\beta = \max(\{0\} \cup \{\delta(l) \mid \overline{l} \in C \text{ and } \delta(l) < \delta\})$$

It is worth noting that when $\beta < \delta - 1$ the algorithm backtracks over several levels, in which case we say that CDCL backjumps to β. Non-chronological backtracking is a major feature of CDCL alongside clause learning as both enable to avoid wasting a significant amount of resources while exploring dead-end regions of the search space. This is clearly showcased in the following example.

Example 2.5. *We consider the same formula $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ in Example 2.1 where $C_1 = x_1 \vee \overline{x_2}$, $C_2 = \overline{x_2} \vee x_3$, $C_3 = \overline{x_3} \vee x_4$, $C_4 = \overline{x_3} \vee \overline{x_4}$ and $C_5 = \overline{x_1} \vee x_2 \vee x_3$. The search tree corresponding to the execution of the CDCL algorithm on ϕ is represented in Figure 2.4. The algorithm performs the same steps as in DPLL until the first conflict is detected by falsifying the clause C_4. As shown in Example 2.4, the clause $C_7 = \{\overline{x_3}\}$ is learned and the algorithm backtracks to the level $\delta = 0$. A sequence of unit propagations follows and a branching on literal x_4 leads to the satisfying assignment $\alpha = \{\overline{x_1}, \overline{x_2}, \overline{x_3}, x_4\}$. Note how the CDCL framework helped to avoid dead-end parts of the search space thus reaching a satisfying assignment more efficiently compared to the search performed by DPLL in Figure 2.1.*

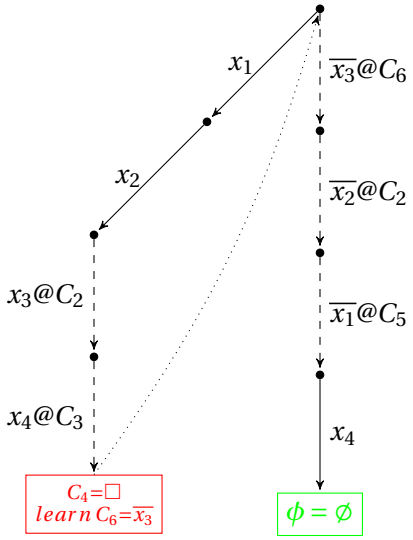


Figure 2.4: Search tree depicting the execution of the CDCL algorithm where branching is made in lexicographic order of the variables. Solid, dashed and dotted arrows represent respectively branching, propagation and backjumping. $l@C$ denotes the propagation of literal l through the clause C.

The basic CDCL procedure is given in Algorithm 2.2 where:

- α and δ represent respectively the current assignment and current decision level.
- **Unit_Propagation**(ϕ, α) applies UP on the formula ϕ and adds the propagated literals to α. It returns $True$ if a conflict is detected, i.e., a clause becomes empty, and $False$ otherwise.
- **Analyse_conflict**(α) analyzes the implication graph to deduce and return a clause C to be learned and a backtrack level β .
- **Backtrack**(ϕ, α, β) undoes all the assignments made within and after the backtrack level β by doing the necessary updates on ϕ and α.
- **Branching_Heuristic**(ϕ, α) chooses the literal (corresponding to a variable and its truth value) to branch on in the search.

Note that in practice, CDCL solvers use an assignment trail in the form of a list of literals associated each to a decision level and, if propagated, to an antecedent clause. This trail is sufficient to perform conflict analysis without maintaining an actual graph structure. Modern solvers also rely on a variety of powerful techniques and heuristics which we present in the next section.

Algorithm 2.2: Conflict Driven Clause Learning (CDCL) Algorithm

Input: a CNF formula ϕ
Output: $True$ if ϕ is satisfiable, $False$ otherwise

```
1:  α ⟵ ∅                                              ▷ Assignment
2:  δ ⟵ 0                                              ▷ Decision Level
3:  while True do
4:      if Unit_Propagation(φ, α) then                 ▷ Inference
5:          if δ = 0 then
6:              return False
7:          C, β ⟵ Analyze_Conflict(φ, α)              ▷ Analysis
8:          φ ⟵ φ ∪ {C}                                ▷ Learning
9:          Backtrack(φ, α, β)                         ▷ Backtracking
10:         δ ⟵ β
11:     else if |α| = |var(φ)| then
12:         break
13:     else
14:         l ⟵ Branching_Heuristic(φ, α)              ▷ Branching
15:         α ⟵ α ∪ {l}
16:         δ ⟵ δ + 1
17: end while
18: return True
```

2.2.1.3 Modern Solvers

Modern SAT solver are able to casually solve instances with millions of variables and/or clauses. This outstanding performance is not only due to the advent of the CDCL framework but also to the variety of techniques and heuristics incorporated in these solvers. Indeed, modern solvers rely on efficient data structures which are tailored for the CDCL algorithm, powerful branching heuristics which cleverly explore the search space, cost-effective handling of the learnt-clauses database to avoid exponential blow-up in the number of clauses, restart mechanisms to deal with heavy-tailed behaviors and enhanced inference techniques to further prune the search space. In this section we present these different techniques with a particular focus on branching heuristics and restarts which are necessary components for our contributions in Chapter 4.

2.2.1.3.1 Lazy Data Structures

One of the major ingredients in the success of modern CDCL solvers is the use of efficient data structures. Indeed, formulas with a high number of clauses, which is constantly evolving through the learning process, warrant the design of suitable data structures to maintain a reasonable cost for recurring CDCL operations such as UP, conflict analysis and backtracking. Traditional data structures for CDCL solvers take the form of adjacency lists [DLL62; SS96; BS97; LA97a]. The clauses are represented as lists of literals and each literal l is associated to its adjacent clauses, i.e., the clauses that contain l. The adjacency

lists of l and $\bar{l}$ enable to access the clauses that are impacted by the assignment of l and, therefore, the number of clauses that need to be checked is significantly reduced. Furthermore, different approaches can be used to identify satisfied, unsatisfied or unit clauses such as simply hiding assigned literals by keeping them in separate lists or by using counters for the number of satisfied/unsatisfied literals to keep track of the status of a clause without having to check all its literals as proposed in the GRASP solver [Mar95; SS96]. Note that adjacency lists are "accurate" data structures as they enable to know exactly the value of each literal in the clause and have the common drawback of associating each literal to a potentially large number of clauses, which often increases as the search proceeds.

Modern data structures opt for a more lazy approach, in which it is not necessary to know the actual state of each literal in the clauses, hence their name. A key feature of these structures is that they mainly need to capture the state of each clause when it is unit to perform UP or when it is unsatisfied to perform conflict analysis. To this end, it is sufficient to track only two literals in each clause as in the Head/Tail (HT) structure introduced in the SATO solver [Zha97]. More specifically, this lazy data structure keeps track of two references in each clause pointing to the head (H) and the tail (T) literal. Initially, H points to the first literal and T to the last literal in the clause. When a literal l pointed by H (resp. T) is assigned, it is replaced by the next unassigned literal on the right (resp. on the left) in the clause. As such, H and T always move towards the middle of the clause. In case the other reference is reached and no unassigned literal can be identified, the clause is declared unit, unsatisfied or satisfied. When the algorithm backtracks, H and T are placed at their previous positions. The behaviour of this data structure is illustrated on the right in Figure 2.5.

The Watched Literals (WL) structure introduced in the Chaff solver [Mos+01], at the heart of modern CDCL solvers, was inspired by its predecessor HT and aimed to remedy its major drawback of updating references during backtrack. To this end, WL does not impose any order relation between the two maintained references. The lack of order between the two watched literals entails that no literal references need to be updated when backtracking. As such, it is also not necessary to keep additional references to save previous positions of watched literals as showcased on the right of Figure 2.5. Many variations of this structure have been introduced in the literature such as Watched Literals (WL) with literal sifting [LM05], Watched List with Conflicts' Counter (WLCC) [Nad02] as well as other optimizations [Gel02; Bie08b; Gen13].

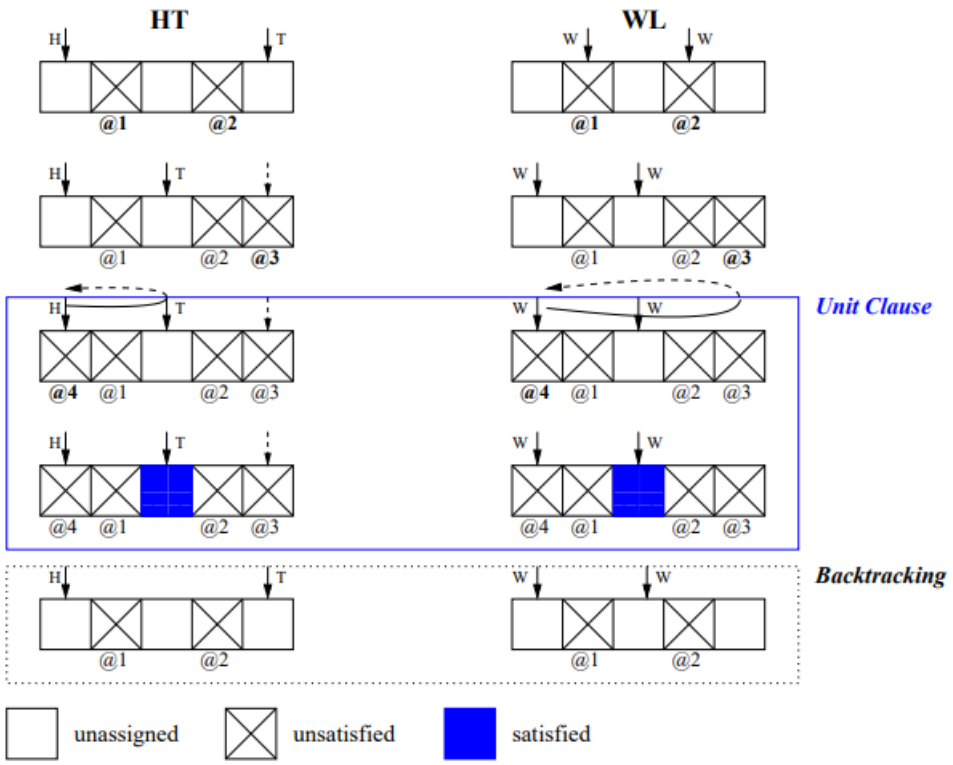


Figure 2.5: Behavior of the Head/Tail (left) and Watched Literals (right) lazy data structures [LM05]. Vertical solid and dashed arrows respectively represent current and saved references. @i denotes that the corresponding literal was the ith assigned literal in the clause.

2.2.1.3.2 Branching Heuristics

Branching heuristics play an important role in modern solvers. Indeed, a good heuristic can highly impact a solver's performance as shown in Example 2.6. There are many heuristics that were designed through the years since the emergence of DPLL and later the CDCL framework. It is important to note that variable branching heuristics and phase branching heuristics, which choose a truth value for the selected variable, can be distinguished. In general, branching heuristics focus on choosing a "good" variable to branch on in the search and can be seen as a scoring function $score : V \longrightarrow \mathbb{R}_+$ on the set of variables $V = var(\phi)$ in the formula, although some exceptions on the nature of this function are possible. The branching is therefore usually done by choosing the variable x with a maximal score, i.e., $x = argmax_{v \in V} score(v)$. Phase heuristics are less common as they were usually integrated with variable branching in the early heuristics whereas practically all modern solvers implement phase saving [PD07a], which forces a variable to be reassigned to its last truth value. As such, we will mainly focus in this section on variable branching heuristics.

Example 2.6. *We consider the same formula* $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ *where* $C_1 = x_1 \vee \overline{x_2}$, $C_2 = \overline{x_2} \vee x_3$, $C_3 = \overline{x_3} \vee x_4$, $C_4 = \overline{x_3} \vee \overline{x_4}$ *and* $C_5 = \overline{x_1} \vee x_2 \vee x_3$ *as in Example 2.1. Note how branching on* $\overline{x_3}$ *would have led to the propagation of* $\overline{x_2}$ *and* $\overline{x_1}$ *respectively through the clauses* C_2 *and* C_5*. Then, branching on literal* x_4 *would lead to a satisfying assignment without encountering any conflict unlike the search trees with lexicographic branching in Figures 2.1 and 2.4.*

Early heuristics for SAT solving were mainly of syntactic (i.e., lexicographic) nature and tended to favor the variables that appeared in clauses of small size. They were also greedy as they made decisions that satisfied immediate goals expected to simplify the problem. Note that these heuristics can be static or dynamic depending on whether they are computed with respect to the input formula or the formula simplified by the current assignment. Hereafter, we present some of the best known traditional syntactic heuristics which were popular in the 90s.

Jeroslow-Wang (JW) [JW90] This heuristics associates to each variable $v \in V$ the following score:

$$score(v) = o(h(v), h(\overline{v}))$$

where:

- $o \in \{max, sum\}$. If $o = max$, we are dealing with the one-sided variant of JW, otherwise we are dealing with the two sided variant.
- $h(l) = \sum_{C \in \phi \text{ s.t. } l \in C} 2^{-|C|}$ and can be used to choose the truth value of the selected variable by comparing $h(v)$ and $h(\overline{v})$.

Böhm's Heuristic [BB92] This heuristic considers a different scoring function $score : V \longrightarrow \mathbb{R}_+^n$ whose output, given a variable v, is a vector $(H_1(v), \ldots, H_n(v))$ such that for $i \in \{1, \ldots, n\}$ we have:

$$H_i(v) = \alpha * \max(h_i(v), h_i(\overline{v})) + \beta * \min(h_i(v), h_i(\overline{v}))$$

where:

- $h_i(l)$ denotes the number of clauses of size i where the literal l appears.
- α and β are two parameters fixed empirically [18].

The heuristic chooses the variable v with maximal vector $score(v)$ under the lexicographic order. Furthermore, v is affected to $True$ if $\sum_{1 \leq i \leq n} h_i(v) \geq \sum_{1 \leq i \leq n} h_i(\overline{v})$, and to $False$ otherwise.

[18] $\alpha = 1$ and $\beta = 2$ in [BB92]

Maximum Occurrences on Minimum Sized Clauses (MOMS) [ZM88; Dub+93; Pre93] This heuristics associates to each variable $v \in V$ the following score:

$$score(v) = h(v) + h(\overline{v}) + \alpha * min(h(v), h(\overline{v}))$$

where:

- $h(l)$ denotes the number of occurrences of literal l in the shortest clauses.
- α is a parameters fixed empirically [19].

Literal Count Heuristics [Sil99] These heuristics implemented in the GRASP solver [Mar95; SS96] associate to each variable $v \in V$ a score as in JW where:

- $o \in \{max, sum\}$. If $o = max$, we are dealing with the Dynamic Largest Individual Sum (DLIS), otherwise we are dealing with Dynamic Largest Combined Sum (DLCS).
- $h(l)$ denotes the number of unresolved clauses in which l appears and can be used to choose the truth value of the selected variable by comparing $h(v)$ and $h(\overline{v})$.

Note that literal count heuristics were also augmented by a random component to decrease their greediness in [Sil99].

Many following heuristics were strongly inspired by those above. Such heuristics were look-ahead by design as they mainly aimed to assess the impact of branching on the search. Although these heuristics may lead to accurate choices, they usually require high resources in term of computation time and storage space. Furthermore, these heuristics mainly targeted random instances, on which they were evaluated. Hereafter, we present the main look-ahead heuristics in the literature.

Look-Ahead MOMS Variations [Fre95; CA96; LA97b] There are several variants of the MOMS heuristic that opted for a more look-ahead approach aiming to evaluate the number of propagations that would be caused by a branching choice. These variations used UP extensively [Fre95; CA96] or relied on composite approaches [LA97b] to evaluate the variables. A slightly different scoring function was introduced in [Fre95] where, given a natural number k [20], a variable $v \in V$ is scored as follows:

$$score(v) = h(v) + h(\overline{v}) + 2^k * h(v) * h(\overline{v})$$

[19] $\alpha = 1.5$ in [Dub+93]

[20] k has a fairly high value initially which can be decreased during the search in [Fre95] whereas it is fixed to 1024 in [CA96; LA97b].

Backbone Heuristics [DD01; DD04] These heuristics aimed to branch on variables which are likely to be in the backbone [21]. To this end, the heuristic in [DD01] for 3-SAT scored each variable $v \in V$ as follows:

$$score(v) = h(v) * h(\overline{v}) \; s.t.$$

$$h(l) = \sum_{(l_1 \vee l_2) \in \mathscr{J}(v)} (2p_1(\overline{l_1}) + p_2(\overline{l_1})) * (2p_1(\overline{l_2}) + p_2(\overline{l_2}))$$

where:

- $\mathscr{J}(v)$ is the set of all binary clauses derived from ternary [22] clauses of the current formula such that each of these binary clauses assigned to False implies v either directly or by virtue of certain unit clauses having the value False.
- For a given literal t, $p_1(t)$ and $p_2(t)$ denote respectively the number of unit and binary clauses in $\phi_{|_t}$ where ϕ is the current formula.

Note that the above estimation is computed on a single level but can be done on multiple levels as specified in [DD01]. This heuristic was also extended for k-SAT with $k \geq 3$ in [DD04].

Recursive Weight Heuristic [MDH10; AF10] This heuristic originally intended for 3-SAT in [MDH10] and extended to k-SAT in [AF10] computes the scores (or weights) of the variables by updating them recursively in each iteration. Given a variable $v \in V$ and an iteration i, the score is defined as follows:

$$score(v) = max(h_i(v), h_i(\overline{v})) \; s.t.$$

$$h_{i+1}(l) = \sum_{(l \vee l_1 \vee l_2) \in \phi} \frac{h_i(\overline{l_1}) * h_i(\overline{l_2})}{\mu_i} + \gamma \sum_{l \vee l_1} \frac{h_i(\overline{l_1})}{\mu_i}$$

where:

- For a literal l, $h_i(l)$ is initialized to 1 (when i=0) and can be used to chose a truth value for the best variable by comparing $h(v)$ and $h(\overline{v})$.
- $\mu = \frac{1}{2n} \sum_{x \in var(\phi)} (h_i(x) + h_i(\overline{x}))$ is the average value used to scale the $h_i(l)$ values at each step.
- γ is an empirically-fixed parameter expressing the relative importance of binary clauses [23].

[21] The backbone of a satisfiable formula is the set of variables having the same truth value in every model [Mon+99; WGS03; Kil+05]. The reader can refer to Section 2.2.3 for more details on structural properties of SAT formulas.

[22] Ternary clauses contain exactly three literals.

[23] $\gamma = 3.3$ in [MDH10]

It is worth noting that this heuristic can be considered as a generalization of the heuristics in [LA97b; Li99; DD01].

The advent of CDCL and the emerging of Artificial Intelligence (AI) in the 2000s led to ever increasing demands in terms of robust and effective heuristics. Furthermore, since choosing the optimal literal to branch on was shown NP-difficult in [Lib00] and solvers were required to deal with larger formulas, modern heuristics were designed in cohesion with the CDCL framework while opting for a look-back approach in which the variables are evaluated through the information accumulated during previous search with minimal consumption of resources. Such heuristics also follow the fail-first principle which states that "to succeed, try first where you are most likely to fail" [HE80].

One particular heuristic which reshaped the SAT heuristics landscape is the Variable State Independent Decaying Sum (VSIDS) which was implemented in the Chaff solver [Mos+01]. After more than two decades since its introduction, VSIDS is still the dominant heuristic used in reference solvers (e.g., MiniSat [ES03a], Glucose [AS09a] and Cadical [Bie17]) and competitive solvers (e.g., MapleCOMSPS [Lia+16c], Maple_LCM [Xia+19a], Cadical [Bie17] and Kissat [Bie+20]) alike, although with some variations. It is worth noting that VSIDS was introduced alongside the Watched Literals (WL) lazy data structure described in Section 2.2.1.3.1, and was therefore adapted to the lack of knowledge on the accurate dynamic size of a clause during the search which renders most of the heuristics described above unusable. Another recent heuristic which seems competitive with VSIDS is the Conflict History-Based (CHB) branching heuristic [Lia+16a]. We introduce below these two major heuristics and their variants as well as some other heuristics in the literature. Note that modern solvers maintain a priority queue [24] in which unassigned variables are sorted with respect to their scores.

Variable State Independent Decaying Sum (VSIDS) [Mos+01] This heuristic introduced in the Chaff solver maintains a score, also called activity, for each variable which is initialized to 0 at the beginning of the search and is used to chose the next branching variable. Two operations are periodically performed on the activity score:

- **Bump:** after each conflict, the score of the variables in the learned clause is incremented, typically by 1
- **Decay:** all the scores are multiplied by a decay factor $0 < f < 1$ periodically, i.e., after m conflicts [25]

Another variant of VSIDS, more commonly used in modern solvers, is the version introduced in MiniSat [ES03a] in which the scores of the variables in all the clauses involved in conflict analysis (i.e., clauses which were resolved including the conflicting clause) are bumped [26]. Furthermore, in this variant, the score decay is performed at each conflict [27].

[24] often implemented as a binary heap [Cor+09]
[25] $f = 0.5$ and $m = 256$ in Chaff
[26] Although this variation is often credited to MiniSAT, it is largely inspired by the branching heuristic of the Berkmin solver which uses VSIDS as an auxiliary heuristic with a similar variation [GN02].
[27] $m = 1$ and $f = 0.95$ in MiniSat

Another implementation feature of VSIDS decay in MiniSat is bumping the variable scores with larger numbers until they reach a certain limit [28] after which they are scaled down.

The Normalised VSIDS (NVIDS) variant, introduced in [Bie08a] and implemented in the Lingeling solver [Bie10b], represents an exponential moving average on how often a variable occurred in antecedents of learned clauses. To this end, given the decay factor f, the score of all the variables are decayed after each conflict including the variables involved in the conflict whose scores are bumped (and decayed) as follows:

$$score(v) = f * score(v) + (1 - f)$$

The Exponential VSIDS (EVSIDS) variant was also introduced in [Bie08a] and aimed to offset the NVSIDS disadvantage of updating all the variable scores at each conflict. Inspired by MiniSAT's sclaing, EVSIDS maintains an exponential increasing score increment $g^{i}(v)$ where $g = 1/f$ and $i(v)$ denotes the conflict index of variable v, i.e., the number of conflicts when a variable was last bumped. Variable scores and the score increment can also be rescored occasionally. As such, only scores of variables that are involved in the conflict need to be updated as follows:

$$score(v) = score(v) + g^{i(v)}$$

Finally, the Average Conflict-Index Decision (ACIDS) variant introduced in [BF15] ensures that the influence of earlier conflicts decreases exponentially with respect to the bumping steps. Given a variable $v \in V$, the score is updated as follows:

$$score(v) = (score(v) + i(v))/2$$

Berkmin's Heuristic [GN02; GN07] This heuristic introduced in the Berkmin solver branches on the unassigned variable in the most recently learned clause that is not yet satisfied by the current assignment. VSIDS is used as an auxiliary heuristic to pick a variable when no such clause exists or to choose one among unassigned variables otherwise. Berkmin introduced the following variation to VSIDS: the scores of all the variables involved in the conflict are bumped with respect to their occurrence in the resolved clauses including the conflicting clause.

Variable Move-To-Front (VMFT) [Rya04] This heuristic implemented in the Siege solver [29] simply moves a subset of the variables in each learned clause to the front of the decision queue. The size of this subset is a small constant m [30] and the variables are positioned arbitrarily at the front of the queue. This heuristic does not necessarily require associating scores to variable. However an efficient implementation is described in [BF15] where variables are simply scored by their conflict index.

[28] the floating point number limit is used in MiniSat
[29] https://www2.cs.sfu.ca/research/groups/CL/software/siege/
[30] $m = 8$ in [Rya04]

Conflict History-Based (CHB) Heuristic [Lia+16a] This heuristic, implemented in the MapleCOMSPS solver [Lia+16c], is based on the Exponential Recency Weighted Average (ERWA) [31] [SB98] and favors the variables involved in recent conflicts as in VSIDS. CHB maintains a score (or activity) for each variable $v \in V$, initially set to 0. The score is updated when a variable is branched on, propagated, or asserted using ERWA as follows:

$$score(v) = (1-\alpha) \times score(v) + \alpha \times r(v)$$

The parameter $0 < \alpha < 1$ is the step-size, initially set to 0.4 and decayed by 10^{-6} after every conflict to a minimum of 0.06. $r(v)$ is the reward value for variable v which can decrease or increase the likelihood of picking v. Higher rewards are given to variables involved in recent conflicts according to the following formula:

$$r(v) = \frac{multiplier}{Conflicts - lastConflict(v) + 1}$$

where $Conflicts$ denotes the number of conflicts that occurred since the beginning of the search and $lastConflict(v)$ is updated to the current value of $Conflicts$ whenever v is present in the clauses involved in conflict analysis. $multiplier$ is set to 1.0 when branching, propagating or asserting the variable that triggered the score update lead to a conflict, otherwise it is set to 0.9. The idea is to give extra rewards for variables producing a conflict.

A recent variant of CHB called the Learning Rate Branching (LRB) heuristic was introduced in [Lia+16b]. This variant uses a more refined reward function which favors the variables that contributed the most in recent conflicts. Whenever a variable is unassigned through backtracking its score is updated using ERWA as in CHB but using a different reward function defined as follows:

$$r(v) = \frac{nbConflicts(v)}{age(v)}$$

where:

- $age(v)$ denotes the number of conflicts encountered in the period between the assignment of a value to v and its unassignment during backtrack.
- $nbConflicts(v)$ denotes the number of conflicts in which the variable v is involved during the same period.

This variant also included a decay component where the scores of unsigned variables are decayed by a factor $0 < f < 1$ after each conflict [32]. The intuition behind the LRB heuristic is

[31] ERWA is originally used in the context of the Multi-Armed Bandit (MAB) problem which will be introduced in Section 2.4

[32] f is set to 0.95 in [Lia+16b] and the authors suggest performing the decays in batch, i.e., multiply the score of a variable a single time by f^k after k conflicts where it remains unassigned instead of performing k multiplications.

that a free variable contributing to most conflicts during the period in which it was assigned a value is likely to help derive new conflicts quickly if it is assigned again.

Distance Heuristic [Xia+19b] This heuristic used in the Maple_LCM solver [Xia+19a] relies on the longest distance between a variable and the conflict in the implication graph to update the score, rather than its simple presence. The score of each variable v in the implication graph, initially set to 0, is updated after each conflict as follows:

$$score(v) = score(v) + \frac{inc}{Dist(v)}$$

where:

- inc is initialized to 1 and decayed after each conflict using a decay factor $0 < f < 1$ [33].
- $Dist(v)$ denotes the number of vertices in the longest path from a node containing v to the conflict node.

Since this heuristic requires more computations to calculate the distances, it is intended to be used at the beginning of the search process when there are very few conflicts [34].

LBD-Based Heuristics [CWX17; CXC19; CMY19] These heuristics use the Literal Block Distance (LBD) measure [35] of the learnt clauses to score the variables. For instance, the heuristic in [CXC19] updates the scores of the variables involved in the conflict as follows:

$$score(v) = score(v) + \frac{1}{LBD} + \frac{1}{BTL}$$

where:

- LBD denotes the literal block distance of the learnt clause.
- BTL denotes the backtrack level.

We finish this section by referring the reader to different studies evaluating and comparing modern branching heuristics [JN08; Lia+15; BF15; Lia+18a].

2.2.1.3.3 Managing Learnt-Clauses Database

Learning all the clauses produced by conflict analysis in CDCL can be impractical as it can lead to an exponential growth of the number of recorded clauses, therefore exhausting all the available memory space. To maintain a learnt-clauses database of reasonable size, modern solvers employ reduction and deletion strategies based on different measures to

[33] f is set to 0.95 in [Xia+19b].
[34] In Maple_LCM, the distance heuristic is only used in the $5 * 10^4$ first conflicts.
[35] The LBD of a learnt clause is the number of distinct decision levels of its literals. The reader can refer to Section 2.2.1.3.3 for more details.

estimate the quality of learnt clauses. The clauses that are deemed irrelevant with respect to those measures are deleted from the learnt-clauses database.

One of the first measures used to evaluate learnt clauses is their size, as in the Grasp solver [Mar95; SS96]. More specifically, GRASP keeps the clauses whose size is less than a given threshold k while discarding larger clauses as soon as the number of unassigned literals in them is greater than one. A different measure, called activity and inspired by the VSIDS branching heuristic in Chaff [Mos+01], was introduced in the Berkmin solver [GN02] and later used in MiniSat [ES03a]. A weight, referred to as activity, is associated with each learnt clause and is bumped and decayed similarly to variables at each conflict. In [AS09b], the authors introduce the Literal Block Distance (LBD) Measure in the Glucose solver, which is defined as the number of distinct decision levels within a learnt clause. In particular, the clauses whose LBD is 2, called glue clauses, contain one variable of the last decision level and are always kept permanently in the database. In recent modern solvers, clauses whose LBD is below a given threshold are usually kept longer or permanently in the learnt-clauses database [Oh16]. Note that the LBD measure can be dynamically updated during the search whenever a learnt clause is used in UP [AS09b; Son16]. Other less commonly used measures were also introduced in [Jab+14; Guo+14; Ans+15a].

The general scheme used in most modern CDCL solver to reduce the learnt-clauses database size is to sort the set of learned clauses with respect to a given measure [36] and then delete half of the learned clauses. It is important to note that the efficiency of most database-management strategies heavily depends on the reduction frequency and on the amount of deleted clauses. Very few other database reduction schemes were designed for CDCL solvers such as the one in [LN17] which attempts to circumvent the problem of choosing a specific amount of clauses to delete at each step. Finally, other minimisation techniques, aiming to reduce the size of learnt clauses, are also extensively used in modern solvers [BKS04; SB09; HJS10b; Luo+17b].

2.2.1.3.4 Restarts

The restart mechanism is an important component of modern CDCL solvers which was introduced to deal with the heavy-tailed phenomena in SAT [Gom+00]. This phenomena is characterized by the non-negligible probability of an exponential increase in solving time when encountering a difficult problem in a specific part of the search space [GSC97; GSK98]. To deal with this phenomena, we can frequently restart the search somewhere else in the search space and thus avoid heavy-tailed behaviors of SAT solvers. At each restart, the solver would undo all the assignments and restart the search anew at the root of the search tree. Note however that some acquired information is usually maintained after a restart such as the learnt clauses and the variable scores. Furthermore, the restarts are performed when a specific cutoff value is reached, usually represented by the number of allowed conflicts per restart. The restart mechanism used in the earlier CDCL solvers was usually coupled with randomization applied to branching heuristics therefore ensuring that different parts of the search space were visited with high probability in each restart [SS96;

[36] or a combination of measures to break ties as in the latest versions of Glucose [AS18]

GSK98; Mos+01; GN02]. Such simple mechanisms yielded considerable improvements on random and real-world instances, including unsatisfiable ones [GSK98; BM00].

To ensure completeness, the cutoff value would be iteratively increased in each restart originally in the form of an arithmetic sequence. However, other strategies are more commonly used in recent modern solvers. One of these well-known strategies is geometric restarts where a geometric sequence is used to increase the base cutoff [Wal99]. Another strategy relies on a different sequence called the Luby sequence [LSZ93]. As defined below, the Luby sequence generates the following terms $1,1,2,1,1,2,4,1,1,2,1,1,2,4,8,\ldots$ which are multiplied by the base cutoff value. Since the cutoff value is usually small [37], using the Luby sequence entails more frequent restarts compared to a geometric one. Most recent modern solvers use Luby restarts as it was shown that it is the best standalone restart policy in [Hua07; HH14]. Other strategies rely on the LBD values of the learnt clauses [AS12; Lia+18b]. For instance, the strategy in [AS12] used in the Glucose solver triggers a restart when the recently produced clauses have high LBDs and is most effective on unsatisfiable instances. More specifically, a restart is performed when the average LBD value in the last X conflicts multiplied by a factor $f \in]0,1[$ is higher then the LBD average of all the deduced learnt clauses.

Definition 2.9 (Luby Sequence [LSZ93])**.** *Given a natural number k, the Luby sequence is defined as follows:*

$$t_i = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1} & \text{if } 2^{k-1} \leq i < 2^k - 1 \end{cases}$$

The study in [Li+20b] investigates the theoretical power of restarts and proves that CDCL augmented with this mechanism can be exponentially faster on a particular class of satisfiable instances. Additional techniques involving the restart mechanism include phase saving [PD07a] and target phases [BF20]. The former simply saves and reassigns the same truth values to variables after restarts while the latter is an extension aiming to increase the likelihood of generating models by saving promising assignments derived by the solver. In [AS12; Oh15; BF19], it was shown that modern solvers could have different behaviors on satisfiable and unsatisfiable instances depending on the chosen restart policy. This observation could be exploited to enhance a solver's performance on either satisfiable or unsatisfiable instances or on both types of instances by using a hybrid strategy. For example, Glucose's restart policy combines LBD-based restarts while blocking fast (Luby) restarts when approaching complete assignments [AS12]. The solvers Cadical and Kissat also implement a hybrid strategy in which they alternate between a focused mode with Glucose-style fast restarts targeting unsatisfiable instances and a stable mode with fewer restarts targeting satisfiable instances [BF20]. Both solvers also use a hybrid branching scheme in which VMTF [Rya04] and EVSIDS [Bie08a] are respectively used in focused and stable modes, in the spirit of the MIRA solver [LSB05]. The solver MapleCOMSPS [Lia+16c] similarly uses a hybrid branching scheme by blocking the restart-cutoff increases to alternate CHB [Lia+16a] (or LRB [Lia+16b]) phases and VSIDS phases in a round-robin fashion.

[37] For instance, MiniSat 2.2.0 implements both geometric and Luby restarts with a default base cutoff value of 100 conflicts and an increase factor of 2 for the geometric sequence.

2.2.1.3.5 Enhanced Inference and Processing

Modern SAT solvers use powerful techniques aiming to infer information from the formula or to simplify it by reducing the number of clauses, literals or variables [Bie12b; WGS13]. Unit propagation (UP), Pure Literals (PL) and clause learning introduced in Sections 2.2.1.1 and 2.2.1.2 are major examples showing how such techniques can dramatically enhance the efficiency of SAT solvers. Similarly, many other techniques, attempting to balance the trade-off between the amount of achieved inference/simplification and the resources invested in them, were introduced in the literature. These techniques can be categorized into preprocessing and inprocessing techniques, applied respectively before and during the search, although many would fall into both categories. In addition, the inference entailed by these different mechanisms should be sound for SAT by ensuring equivalence [38] or a less constrained form of soundness defined below, called equisatisfiability. It is worth noting that using such techniques became popular with the success of the SatELite preprocessor [39] [EB05] which would later be integrated into MiniSat 2. Following MiniSat's footsteps, recent modern solvers also integrate a variety of techniques to optimize their performance.

Definition 2.10 (Equisatisfiability)**.** *Let ϕ and ϕ' be two formulas. ϕ is equisatisfiable to ϕ' if we have: ϕ satisfiable if and only if ϕ' satisfiable.*

A well-known generic simplification technique is the elimination of redundant clauses [Fou+07; HS09]. Intuitively, a redundant clause represents information that can be derived from the rest of the formula. Eliminating such clauses therefore produces a smaller size equivalent formula. In its simpler form, this technique would consist in deleting duplicated, tautological, or subsumed clauses. However, more complex forms exist such as self-subsuming resolution [EB05] which eliminates clauses subsumed by resolvents obtained through the application of the resolution rule [40] [Rob65; HJB10].

Another redundancy-based technique is the elimination of blocked clauses [JBH10; Bal+14; Kie+18]. Informally, a clause C is blocked if it contains a literal l such that all possible resolvents derived from C with another clause in the formula by resolution on $var(l)$ are tautologies [Kul99]. This technique generalizes failed-literal probing [Fre95; Le 01] which is extensively used in SAT solvers as it can enhance the power of UP in inferring literals that must be satisfied in the current formula. More specifically, when UP is not applicable, we can check if assigning a literal l leads to a conflict, in which case the literal $\overline{l}$ should be satisfied and can be propagated in the current formula. Other extensions of this technique include clause vivification [PHS08; Li+20a] and distillation [JS05; HS07].

Bounded variable elimination [EB05; SP05; RH21] uses the original Davis-Putnam (DP) procedure in [DP60] to remove variables from the formula. This technique consists in simply applying the resolution rule on all the clauses opposed on a given variable v to derive all the non-tautological resolvents while deleting all the clauses containing v. To prevent exponential blow-up in the number of clauses, variables are only eliminated if the number of derived clauses is less than the number of deleted clauses. Finally, to be

[38] refer to Definition 1.17
[39] `http://minisat.se/SatELite.html`
[40] refer to Definition 2.7

exhaustive, we mention some other techniques related with variable elimination such as the detection of functional dependencies between variables [Ost+02; Gré+05; OP09; Ise20] or the elimination of hidden literals (i.e., implying other literals in the same clause) [HJB11].

2.2.1.4 Other methods

In this section, we overview other complete algorithms for SAT solving. One particular family of algorithms in the literature relies on parallel solving [BS18]. Parallel solvers for SAT can be divided into two main categories: portfolio and divide-and-conquer algorithms. The underlying idea behind the portfolio approach is to rely on multiple CDCL-based SAT solvers equipped with different strategies and settings to solve an instance. Indeed, a standalone solver (or solver instance with a specific setting) might perform well on a given set of instances while struggling or even drastically failing on another. Since it is difficult to predict a solver's performance on a specific instance, launching many competitive solvers in parallel on different processors to deal with the same instance may lead to more efficient solving. Once a solver terminates, its result is reported and all other solvers are terminated. State-of-the-art solvers based on this approach include ManySAT [HJS10a], Glucose-Syrup [AS14], HordeSAT [BSS15] and Plingeling [Bie10a; Bie+20] among others.

The second category of algorithms, i.e., divide-and-conquer, splits the search space into partitions which are solved in parallel. Such methods have been applied in the 1990s on the DPLL framework [BS96; ZBH96]. A recent known variant paradigm of divide-and-conquer, which can be applied on CDCL, is the cube-and-conquer algorithm [Heu+12]. The problem is divided into many sections called cubes through a look-ahead solver. Intuitively, cubes are conjunctions of subsets of variables of the original formula that can be solved independently by CDCL solvers. The problem is then tackled using the cubes to guide the search. Solvers that fall within the divide-and-conquer category include Treengeling [Bie12a], MapleAmpharos [Nej+17] and P-CLONE-FLIPS [Le +19], which is based on the Painless framework [Le +17]. Finally, it is worth noting that state-of-the-art solvers in both concurrent categories rely on different clause-sharing techniques [Guo+10; Laz+12; HJS12; AS14; Val+20; PSM21].

Other complete methods for SAT solving include hybrid algorithms which boost DPLL/CDCL solvers through incomplete local search methods which will be presented in the following section. For example, the DPLL framework is augmented with a call to a local search algorithm at each node in [MSG98; Hab+02]. A local search solver is also called to incrementally identify subformulas which are solved by CDCL in [LM08]. More recently, the solvers CaDiCaL and Kissat call a local search solver to produce a promising assignment used for target phases [BF20]. A deeper cooperation between CDCL and local search is proposed in [CZ21b; Cai+22] to improve target phases, rephasing and branching. Finally, other solving paradigms can be used to solve SAT although this is not common as satisfiability is a powerful formalism with highly efficient dedicated solvers. Nevertheless, one can exploit more generic formalisms with more powerful expressiveness and inference such as the recent work in [AL21] which relies on reduction from SAT to Max-2-SAT [41].

[41] refer to Section 3

2.2.2 Incomplete Methods

Unlike complete solvers which are based on exhaustive branching and backtracking search, incomplete methods are mainly based on Stochastic Local Search (SLS) [Gu92]. For some problems and specifically for randomly generated satisfiable instances, incomplete methods for SAT can significantly outperform DPLL-based methods. In this section, we first introduce the general outline of an SLS algorithm for SAT. Then, we present the two major SLS architectures in the literature, namely GSAT [SLM92] and WalkSAT [SKC94]. Furthermore, we introduce the main metaheuristics used to enhance the performance of modern local search SAT solvers. Finally, to be exhaustive, we briefly overview other less common incomplete methods.

2.2.2.1 Stochastic Local Search

Stochastic Local Search (SLS) methods for SAT [Gu92; SLM92], whose outline is described in Algorithm 2.3, seek to improve an initial random assignment of the formula with respect to a cost function through a sequence of simple steps, called flips, which aim to locally repair the current assignment. The cost (or objective/evaluation function), defined below, is used to evaluate each assignment α and is usually set to the number of clauses falsified by α. As such, SLS algorithms for SAT aim to optimize the cost of an assignment α until it reaches a global optimum, defined below. Clearly, in case the global optimum $cost_\alpha(\phi) = 0$ is achieved, then α is a satisfying assignment of the formula ϕ and is returned by the algorithm (lines 4-5).

Definition 2.11 (Cost)**.** *Let ϕ be a CNF formula and α be an assignment of ϕ. The cost of α is defined as follows:*

$$cost_\alpha(\phi) = |\{C \in \phi \,|\, C|_\alpha = \square\}|$$

Definition 2.12 (Global Optimum)**.** *Let ϕ be a CNF formula, $\mathscr{A}$ be the set of all possible complete assignments of ϕ and α be an assignment in $\mathscr{A}$. $cost_\alpha(\phi)$ is a global optimum of the cost function if we have:*

$$\forall \alpha' \in \mathscr{A}, cost_\alpha(\phi) \leq cost_{\alpha'}(\phi)$$

To optimize the current assignment, SLS goes through an intensification phase in which only neighboring assignments can be visited in each step. The set of neighbors of an assignment, defined below, consists of the assignments that differ in exactly one literal. The assignment to be visited in this neighborhood is determined through the choice of a variable, whose truth value is flipped to obtain the new assignment (lines 7-8).

Definition 2.13 (Assignment Neighborhood)**.** *Let ϕ be a CNF formula and $\mathscr{A}$ be the set of all possible complete assignments of ϕ. The neighborhood of an assignment $\alpha \in \mathscr{A}$ is defined as follows:*

$$\mathscr{N}(\alpha) = \{\alpha' \in \mathscr{A} \,|\, |\alpha \cap \alpha'| = |\alpha| - 1\}$$

Algorithm 2.3: Stochastic Local Search for SAT

Input: a CNF formula ϕ, two natural numbers $maxTries$ and $maxSteps$
Output: a satisfying assignment α of ϕ, UNKNOWN otherwise

```
1:  for i = 1 to maxTries do
2:      α ⟵ random complete assignment of φ
3:      for j = 1 to maxSteps do
4:          if α satisfies φ then
5:              return α
6:          else
7:              x ⟵ Choose_Variable(φ, α)
8:              α ⟵ α with truth value of x flipped
9:      end for
10: end for
11: return UNKNOWN
```

Definition 2.14 (Flip). *Let ϕ be a CNF formula, α be an assignment of ϕ and v be a variable of ϕ. A flip step on v produces the assignment $\alpha' \in \mathcal{N}(\alpha)$ defined as follows:*

$$\alpha' = (\alpha \setminus \{l\}) \cup \{\bar{l}\}$$

where l is the literal in α s.t. $var(l) = v$.

Note that the intensification phase could lead to a local optimum. Informally, a local optimum corresponds to an optimal cost within a restricted region of the search space such that visiting neighboring assignments can only degrade the cost function. To avoid getting stuck in local optima, SLS goes through a diversification phase by restarting the search to visit other parts of the search space (lines 1-2) or by diversifying variable choices. Ensuring a good trade-off between intensification and diversification is necessary in devising efficient SLS algorithms. It is worth mentioning that random restarts in SLS inspired the restart mechanism used in modern complete solvers introduced in Section 2.2.1.3.4.

Definition 2.15 (Local Optimum). *Let ϕ be a CNF and α be an assignment of ϕ. $cost_\alpha(\phi)$ is a local optimum of the cost function if we have:*

$$\forall \alpha' \in \mathcal{N}(\alpha), cost_\alpha(\phi) \leq cost_{\alpha'}(\phi)$$

SLS algorithms mainly differ in the way they choose variables to be flipped. In the next sections, we present the two main SLS architectures for SAT then we overview some well-known metaheuristics that are incorporated into SLS algorithms.

2.2.2.2 SLS Architectures

Two major SLS architectures were introduced for SAT, namely GSAT and WalkSAT. These architectures are in fact basic SLS algorithms which are at the heart of every SLS solver. The GSAT algorithm [SLM92], introduced earlier in 1992, tries to minimise the number

of unsatisfied clauses by a greedy descent in the space of variable assignments. Variable selection in GSAT is based on the score of a variable v under the current assignment α, which is defined as follows:

$$score(v) = break(v) - make(v)$$

where $break(v)$ and $make(v)$ denote the number of clauses unsatisfied respectively by α and the assignment obtained by flipping v in α. In each local search step, one of the variables with maximal score is flipped. Random selection with uniform distribution is used when there are several variables with maximal score. The key to efficiently implementing GSAT is to evaluate the complete set of scores only once at the beginning of each try, and then after each flip to update only the scores of the variables which were possibly affected by the flipped variable. A well-known variant of this algorithm with more pronounced diversification in variable selection aiming to avoid getting stuck in local minima is GSAT with random walk (GWSAT) [SKC94]. As its name suggests, GWSAT augments the variable selection heuristic in GSAT with a random walk step. In such a step, a currently unsatisfied clause C is first randomly selected. Then, one of the variables appearing in C is flipped, thus effectively forcing C to become satisfied. The basic idea of GWSAT is to decide at each local search step with a fixed probability wp, called walk probability or noise setting, whether to perform a standard GSAT step or a random walk step. Many SLS algorithms and/or solvers for SAT are based on the GSAT architecture such as H(W)SAT [GW93; GWH95], G(W)SAT with Tabu search [MSG97; SSS97], Sparrow [BF10], G^2SAT [LH05b], TNM [LWL12], sattime [LL12], gNovelty+ [Ngh+08] as well as the configuration-checking-based algorithms [CS11; LSC12; CS12; CS13b; Luo+14; CLS15a].

The WalkSAT architecture [SKC94] is based on a two-stage variable-selection process which randomly chooses a currently unsatisfied clause C then selects one of its variables, as in the the random walk step of GWSAT. However, WalkSAT has two major differences. Firstly, the scoring function used by WalkSAT considers only the break values, i.e., the score of a variable v is simply $break(v)$. Secondly, if there exists a variable v in C such that $break(v) = 0$, it is immediately selected. Indeed, flipping this variable would satisfy the clause C without falsifying another clause thus leading to a zero-damage flip. If no such variable exists, WalkSAT selects the variable having minimal score value with a probability wp similarly to GWSAT. Furthermore, in the remaining cases, a random walk flip is performed in which a variable in C is randomly chosen to be flipped. Major algorithms/solvers based on WalkSAT include Novelty and its variants [MSK97; Hoo99; LH05b; Ngh+08; AHT17] WalkSATlm [CSL13; CLS15b] and ProbSAT [BS12] among others. Finally, we refer the reader to the study in [HS00] which includes an empirical evaluation of the standard algorithms mentioned above.

2.2.2.3 Metaheuristics

Like complete solvers, modern local search solvers for SAT incorporate a variety of techniques to enhance their performance. In this section, dedicated to these different techniques, we start by presenting some well-known metaheuristics relying on information

acquired through the search. Then, we present clause weighting schemes employed in dynamic local search algorithms for SAT. Finally, we describe some search mechanisms used to adaptively set different parameters in SLS solvers.

2.2.2.3.1 History-Based Local Search

Recent SLS algorithms often include techniques that rely on information acquired through the search. One particular information that is used in a variety of algorithms is variable age, i.e., the number of steps since a variable has been flipped. For example, the Tabu mechanism [Glo89] which was combined with both GSAT and WalkSAT in [MSG97; SSS97; MSK97] uses this information to forbid reversing the effect of a flip move. More specifically, a variable cannot be flipped if its age is less than a given parameter, called Tabu tenure. In WalkSAT, if all the variables in the chosen unsatisfied clause are tabooed, another clause is chosen. If no such clause exists, the Tabu mechanism is temporarily ignored. The underlying idea behind this mechanism is to avoid the cycling problem in local search where candidate assignments which have been explored recently are constantly revisited again.

The HSAT algorithm [GW93], based on GSAT, uses variable age to break ties. When several variables with maximal score are identified, HSAT selects the oldest flipped variable. The intuition behind this algorithm is that some relevant variables which are often eligible to be chosen may not get flipped thus causing the search to stagnate. Breaking ties with respect to age also helps to alleviate the cycling problem. HSAT was also extended with GWSAT's random walk mechanism in [GWH95].

The Novelty algorithm [MSK97], a well-known variant of WalkSAT, also keeps track of the age of all flipped variables. After randomly choosing an unsatisfied clause as in WalkSAT, Novelty selects the best variable with respect to the break score only if it does not have minimum age. Otherwise, the second best variable is chosen with probability wp. Similarly to the Tabu mechanism, the intuition behind Novelty is to avoid repeatedly flipping the same variable back and forth. Many variations were introduced to Novelty. For example, the R-Novelty algorithm [MSK97] takes into account the difference in scores between the best and second best variables to choose a relevant one. Other variants include Novelty+ [Hoo99] and Novelty++ [LH05b] among others [Ngh+08; AHT17]. Given a randomly selected unsatisfiable clause C, Novelty+ simply picks a random variable in C with probability wp' while Novelty++ picks the variable with minimal age in C and both use basic Novelty as a sub-procedure with probability $1 - wp'$ otherwise.

Configuration Checking (CC), initially introduced in [CS11], is a powerful technique where a variable is tabooed with respect to the age of its neighboring variables. The neighborhood $\mathcal{N}(v)$ of a given variable v consists of all the variables which occur in at least one of the clauses containing v. The notion of configuration, formally defined below, is thus introduced for a variable by considering the restriction of an assignment α under its neighbors. The underlying idea of CC is to forbid flipping variables whose configurations has not been changed since their last flip. Intuitively, a variable can only be flipped if its age is higher than at least one of its neighbors'.

Definition 2.16 (Variable Neighborhood in CC [CS11]). *Let ϕ be a CNF formula, and v be a variable in ϕ. The neighborhood of v is defined as follows:*

$$\mathcal{N}(v) = \{v' \in var(\phi) \setminus \{v\} \mid \exists c \in \phi \text{ s.t. } v, v' \text{ occur in } c\}$$

Definition 2.17 (Variable Configuration in CC [CS11]). *Let ϕ be a CNF formula, α be a complete assignment of ϕ and v be a variable in $var(\phi)$. The configuration of v is a vector $CC_v \in \{0,1\}^{|\mathcal{N}(v)|}$ consisting of the truth values of all variables in $\mathcal{N}(v)$ under assignment α.*

Many variations and extensions were introduced for the CC mechanism [LSC12; CS12; CS13b; Luo+14; CLS15a; AHT17]. The work in [LSC12] introduces Quantitative Configuration Checking (QCC) with variations on the notions of neighborhood and configuration. As defined below, the configuration is based on the state of the clauses (satisfied or not) with respect to an assignment. The same work also uses the quantity of variations within the configuration, i.e., the number of times the configuration changes smoothed over time, to break ties when choosing a variable.

Definition 2.18 (Variable Neighborhood in QCC [LSC12]). *Let ϕ be a CNF formula, and v be a variable in ϕ. The neighborhood of v is defined as follows:*

$$\mathcal{N}(v) = \{C \in \phi \mid v \text{ occurs in } C\}$$

Definition 2.19 (Variable Configuration in QCC [LSC12]). *Let ϕ be a CNF formula, α be a complete assignment of ϕ and v be a variable in ϕ. The configuration of v is a vector $CC_v \in \{0,1\}^{\mathcal{N}(v)}$ consisting of the states (satisfied or falsified) of all the clauses in $\mathcal{N}(v)$ under assignment α.*

The Configuration Checking with Aspiration (CCA) algorithm described in [CS12] extends CC with a greedy aspiration mechanism where variables whose flips can bring a considerable benefit have a chance to be selected even if they do not satisfy the CC criterion. In [CS13b], CCA is extended with a subscore property used to break ties on variable scores and mainly targeting k-CNF formulas with $k > 3$. The Double Configuration Checking with Aspiration (DCCA) strategy introduced in [Luo+14] employs a double CC scheme combining the two different notions of configuration introduced above based on variable and clause neighborhoods. The CC strategy was also incorporated within the Novelty framework in [AHT17]. Finally, most recent state-of-the-art SLS solvers implement the CC strategy such as Swcc and its variants [CS11; LSC12; CS12], (D)CCASat [CS13b; Luo+14], Cscore-SAT [CS13a; CLS14] , CCAnr [CLS15a] and Ncca+ [HTA13; AHT17]. In particular, the solver CCAnr [42] based on the CCA strategy augmented with the preprocessor [43] CP3 [Man12] shows better performance on structured instances compared to other SLS solvers.

[42] http://lcs.ios.ac.cn/~caisw/SAT.html
[43] refer to Section 2.2.1.3.5 for processing techniques

2.2.2.3.2 Dynamic Local Search

Dynamic Local Search (DLS) schemes for SAT, also called clause weighting schemes, associate a weight (or penalty) to each clause in the formula. Intuitively, the weight represents the difficulty of a clause as a constraint in the formula. Clause weights have a fixed initial weight, typically 1, and are dynamically adjusted during the search so as to penalize clauses which are difficult to satisfy within local minima. Unlike previously discussed algorithms and techniques which mainly rely on break and/or make scores of variables, clause weights are used to guide the search in DLS. More specifically, the variable which produces the greatest reduction in the sum of weights of unsatisfied clauses is flipped as first specified in [Mor93]. DLS algorithms differ primarily in the schemes used to adjust the clause weights. Two major schemes have emerged in the literature: additive and multiplicative weighting.

The first clause weighting methods [Mor93; SK93] relied on a purely additive scheme where the weights of unsatisfied clauses are incremented whenever the search encounters a local minimum, i.e., the objective function cannot be improved, or simply at the end of each try. Such methods allowed unrestricted weight growth during the search. In subsequent work, the Discrete Lagrangian Method (DLM) [WW00] additionally decremented clause weights after a fixed number of increases while allowing zero-damage flips (called flat moves in [WW00]). One of the best DLS schemes is the Pure Additive Weighting Scheme (PAWS) [Tho+04; Tho06] used in current state-of-the-art solvers. The major feature of PAWS is the use of a smoothing probability sp to decrement the clause weights during the search, otherwise the weights are incremented as in DLM.

Multiplicative weighting emerged with the Smoothed Descent and Flood (SDF) method for breaking ties between zero-damage flips. In the case where no improving flips are possible, SDF increases weights of unsatisfied clauses and then smooths clause weights so that the greatest cost difference between any two flips remains constant. Other works include the exponentiated subgradient algorithm in [SSH01] which inspired the well-known Scaling And Probabilistic Smoothing (SAPS) scheme [HTH02]. The additional characteristics of SAPS compared to the work in [SSH01] are the periodic smoothing of weights with respect to a smoothing probability, which was later applied in PAWS, and their increase only for unsatisfied clauses within local minima.

The study in [Tho06] provides a more detailed overview of clause weighting schemes. It is worth noting that clause weighting, mainly in the form of the PAWS scheme, has been combined with state-of-the-art techniques including Configuration Checking (CC) within recent SLS solvers such as CCAsat [CS13b], CCAnr [CLS15a] or Ncca+ [AHT17]. Other works related to DLS also include switching between non-weighting, clause weighting, and variable weighting schemes as described in [WLZ08] or efficiently handling tie-breaking situations encountered in the landscapes of additive weighting [FT05].

2.2.2.3.3 Adaptive Local Search

State-of-the-art SLS solvers often rely on different parameters which can critically impact their performance [MSK97; Hoo02]. Although such parameters are usually fixed empirically, many studies introduced and investigated schemes in which their values are adaptively set

during the search. For example, an adaptive noise mechanism for WalkSAT was introduced in [Hoo02]. Through this mechanism, the walk probability is increased during intensification phases to further guide the search with more greedy steps whereas it is decreased when there is stagnation in the cost function for a set period of search steps, thus further diversifying the search to escape local minima. This mechanism can be easily applied to other algorithms with a similar parameter such as the noise setting in GWSAT and can be combined with other metaheuristics such as Dynamic Local Search (DLS) as in [Ngh+08]. In particular, in the context of DLS and particularly the Scaling and Probabilistic Smoothing (SAPS) multiplicative scheme [HTH02], the authors propose to adaptively tune the smoothing probability parameter during the search similarly to [Hoo02]. Such a mechanism can also be applied to the PAWS scheme as showcased in [Tho06].

2.2.2.4 Other Methods

One of the major challenges in terms of incomplete methods for SAT is to devise algorithms which are efficient on industrial instances. Indeed, SLS solvers in the literature lag far behind competitive CDCL-based SAT solvers in this regard, including the configuration-checking-based local search solver CCAnr [CLS15a] which is tailor-made for real-world instances. One research area that may circumvent this problem is devising hybrid algorithms which take advantage of the power of local search and CDCL. Note that many studies have incorporated techniques used in complete solvers into SLS. For instance, Unit Propagation (UP) was used to simplify the formula before the search in [LL12; CLS15a] or to generate a relevant initial assignment at the beginning of each try in [Cai+21]. UP was also used during search steps [LSB04; HK05] and within local minima [GH11]. Many attempts to devise efficient fully hybrid incomplete solvers were made in the literature. Examples of such solvers include hybridGM [BHG09] and SATHYS [Aud+10] which call a CDCL solver in local minima. Other incomplete methods for SAT also include parallel local search [Rol02; AH11; BS18], simulated annealing [Spe93; Bee+94] and evolutionary algorithms [GV98; Lov15; Fu+18].

2.2.3 Behind the Success of SAT Solving

The theoretical difficulty of SAT has been established since the 70s as it was the first problem to be proven NP-complete in [Coo71]. Indeed, no polynomial algorithm has been devised for the SAT problem and it is highly plausible that no such algorithm exists. However, modern solvers are extremely efficient and manage to outstandingly solve formulas with millions of variables and clauses. Many studies have attempted to investigate this conflicting gap between the theoretical difficulty of SAT and the remarkable practical performance of modern solvers [AMM22a].

One of the first research directions aiming to shed the light on instance hardness is phase transitions [MSL92; LT92; GW94a; FB+99; XL99; ZPV10; BH22]. These transitions are distinguished by a sharp satisfiability threshold r_k for random k-SAT formulas where $k \geq 2$. This critical value represents a ratio of the number of clauses to the number of variables which characterizes an easy-hard-easy pattern in the median difficulty of the problems, with the hardest problems being close to r_k. It was shown in [AM02; CP16] that for large values of

k, the asymptotic order of the phase transitions threshold is $r_k \sim (ln2).2^k$. Note that phase transitions are mainly studied on purely-random formulas although a recent study [ABL19] investigated this feature for random formulas based on realistic models [ABL09c; GL17].

A second promising research area in this context is identifying traceable classes for the SAT problem such as 2-SAT [Kro67] and Horn-SAT [DG84] [44] among many others [Lew78; CH91; BHS94; Sch+95]. The notion of Backdoor introduced in [WGS03] generalizes this concept with the underlying motivation that assigning a certain subset of the variables in the formula may produce a simplified formula within a tractable class and therefore which is easy to solve. Formally, a strong [45] backdoor of a SAT instance is the set of variables which, however they are assigned, give a simplified formula which can be solved in polynomial time. Note that the polynomial sub-solver can be defined algorithmically, e.g., Unit Propagation (UP), or syntactically using an established tractable class. The size of strong backdoors is highly related to the hardness of SAT instances [LM04; Kil+05]. The notion of backdoor was extended in the context of clause learning in [DGS09], and it was shown that the smallest backdoors which take into account learning and branching can be exponentially smaller than traditional backdoors thus giving some insights on the additional power of clause learning in CDCL. Other extensions include deletion [NRS06], pseudo [08], recursive [MSV21] and probabilistic [Sem+22] backdoors. Note that, under the assumption P≠ NP, identifying backdoors is NP-hard [DGS07]. Nevertheless, many attempts were made to efficiently identify (small) backdoors in the literature [Kil+05; Ost+06; KKS08; LB11; GK22] including Fixed-Parameter Tractable (FPT) algorithms [46] [NRS04; GS12a; GS12b; LPR22].

Another hidden structure which is often associated with backdoors is the notion of backbones [Mon+99; WGS03; Kil+05]. Formally, the backbone of a satisfiable formula is the set of variables which have the same truth value in every model. Many methods were devised to compute backbones in the literature [MJL10; Zhu+11; JLM15; Zha+18] although it is established as an NP-hard problem [Mon+99; Kil+05]. The study in [Kil+05] highlights the correlation between the size of backbones (and strong backdoors) and problem hardness while the empirical evaluations carried out in [MJL10; JLM15] show that real-world instances harbor large backbone sets. Many studies also rely on this notion to boost SAT solving both for complete and incomplete algorithms [DD01; DD04; Zha04; AMM22b]. The notion of variable entropy introduced in [CS18] can be seen as a generalization of backbones. Intuitively, this property approximates the freedom of assigning variables in satisfiable formulas and have shown promising results in predicting problem hardness.

A different structural feature, broadly investigated in the literature, is the potential functional dependencies between variables which are usually recovered and exploited for preprocessing [Ost+02; Gré+05; OP09]. Intuitively, a dependency takes the form of a gate where an output variable v is semantically linked to a set of input variables $\{v_1, v_2, \ldots, v_n\}$ in the formula through connectors. Formally, the gate can be expressed in the form of an equation $v = f(v_1, v_2, \ldots, v_n)$ where $f \in \{\wedge, \vee, \Leftrightarrow\}$, although more generic definitions of f as

[44] Refer to Section 2.1

[45] in contrast to weak backdoors which are restricted to satisfiable formulas

[46] In parameterized complexity [DF13], a problem is called Fixed-Parameter Tractable (FPT) if there exists an algorithm deciding the problem in time $f(k).n^c$ where n is the input size, k is the parameter, f is a computable function and c is a constant.

a boolean function are possible [Ise20]. Gate structure is mainly a result of explicit structural encodings [Tse83; PG86] but gates can be more implicitly nested within formulas [IMS15; Ise20].

Structural properties of graphical representations of SAT formulas have also been widely studied in the literature. This trend started with the study in [Wal99] where the graphs associated with different combinatorial problem were shown to have small-world topology inducing heavy-tailed distribution, i.e., nodes are highly clustered with small path length between them. Subsequent studies for SAT focused on analyzing the structural properties of the Variable Incidence Graph (VIG)[47] and/or the Clause-Variable Incidence Graph (CVIG), defined below. In [ABL09d], the authors show that, in contrast to random formulas, the CVIGs of industrial instances seem to exhibit scale-free structure where the arity of nodes follow a power-law distribution $P(a) \sim a^{-\gamma}$ (with γ around 3). The same authors relied on this structure to generate random instances with more realistic (i.e., industrial-like) features [ABL09c; ABL22].

Definition 2.20 (Primal Graph)**.** *Let ϕ be a CNF formula. The primal graph $G = (V, E)$ is defined as follows:*

- $V = var(\phi)$
- $(v_1, v_2) \in E$ *iff* $\exists C \in \phi$ *s.t.* v_1 *and* v_2 *occur in* C

Definition 2.21 (Clause-Variable Incidence Graph)**.** *Let ϕ be a CNF formula. The Clause-Variable Incidence Graph (CVIG) is a bipartite graph $G = (V \cup \Gamma, E)$ defined as follows:*

- $V = var(\phi)$ *and* $\Gamma = \phi$
- $(v, C) \in E$ *iff* $v \in var(\phi)$ *occurs in* $C \in \phi$

Another property which gained particular interest in recent years is the modularity of SAT instances [AL11; AGL12; Ans+19; Li+21b]. More specifically, a modular graph exhibits independent partitions of its vertices, called communities, which are highly constrained as they internally harbor many edges. On the other hand, the communities are externally linked through very few edges. It is established that the VIG and CVIG graphs [48] of industrial formulas are highly modular in contrast to random formulas, as showcased in Figure 2.6. Modularity is shown to be correlated to the runtime of CDCL solvers in [New+14; New+15] but the study in [MFS16] indicates that industrial instances may have some other relevant structure not captured by this model. Community structure was also used to detect relevant learnt clauses in [Ans+15b] and to generate random SAT instances in [GL15; GL16]. Other (C)VIG-based structural features studied in the literature include treewidth [49], variable

[47] also called primal graph

[48] These graphs are weighted so as to ensure that the sum of weights of edges generated by a given clause always equals one.

[49] This measure intuitively represents the closeness of a graph to a tree. Note however that the attempt in [Mat11] was not successful as the authors concluded that the treewidth of the primal graph is not a good indicator for the hardness of SAT instances.

centrality [KS12; JM17; JM18], self-similarity [50] [Ans+14] and entropy [51] [ZXZ21].

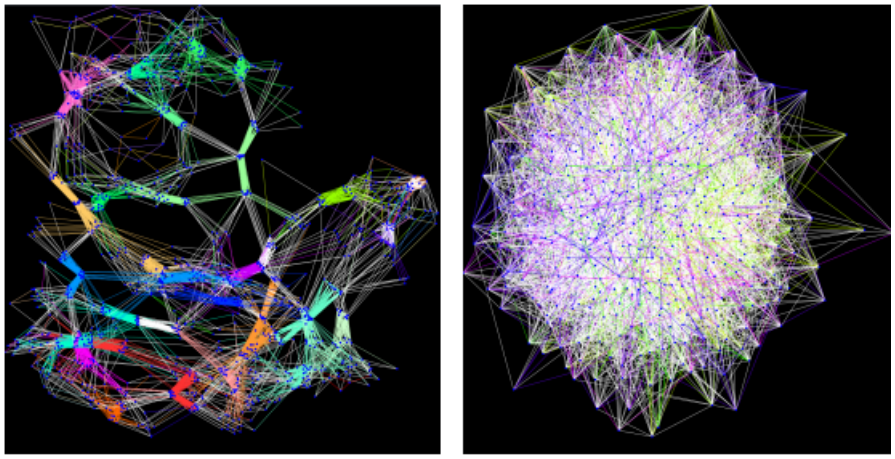


Figure 2.6: Community structure of industrial (left) and random (right) SAT instances [New+15]

2.3 Proofs and Certificates for SAT

In this section, we are interested in certificates and proof systems for the SAT problem. We start by defining some necessary notions and overviewing proof systems for SAT. Then, we focus on resolution [Rob65] as a proof system for SAT, which is one of the first and most studied refutation systems in the literature. Finally, we present some known classes of resolution which will be used in our contributions in Chapter 6.

2.3.1 Proof Systems for SAT

A SAT solver tasked to solve a given formula can easily provide a short certificate if the formula is satisfiable. Indeed, this certificate of satisfiability can simply take the form of a satisfying assignment α. Such a certificate not only has a reasonable size with respect to the size of the input formula but can also be easily verified by checking that all the clauses in the formula are satisfied by α. In the context of decision problems and particularly the SAT problem, the need of certifying the solution can arise both for positive and negative instances. When a formula is unsatisfiable, a certificate is mainly provided through the lens of proof systems. Intuitively, if a contradiction can be inferred from the formula using sound inference steps for SAT, then the formula can never be satisfied and the sequence of deductions form a proof that can be used as certificate of unsatisfiability.

Formally, a proof system for SAT is a set of inference rules. An inference rule is defined by its antecedents and conclusions, both usually expressed in the form of a set of clauses

[50] This feature intuitively means that rescaling a graph by replacing groups of nodes with single nodes does not change its structure.

[51] This feature is closely related to the variable entropy measure in [CS18].

(although other forms are possible), such that the conclusions can be added if the antecedent clauses are within the formula. An example of a simple inference rule for SAT, called weakening, is given below. We denote $\phi \vdash_S \phi'$ (resp. $\phi \vdash_S C$) if the formula ϕ' (resp. the clause C) can be derived from ϕ in the system S. To guarantee correctness of inference induced by the different rules, their soundness must be ensured. More specifically, an inference rule is sound for SAT if it preserves SAT equivalence, defined below.

Definition 2.22 (Weakening)**.** *Given two clauses C_1 and C_2, the weakening rule is defined as follows:*

$$\frac{C_1}{C_1 \vee C_2}$$

Definition 2.23 (SAT Equivalence [52])**.** *Let ϕ and ϕ' be two formulas. ϕ and ϕ' are equivalent (for SAT), denoted $\phi \equiv \phi'$, if for any assignment α of ϕ, α is a model of ϕ iff α is a model of ϕ'.*

Definition 2.24 (Soundness)**.** *A proof system S is sound if, for any formulas ϕ and ϕ', $\phi \vdash_S \phi'$ implies $\phi \equiv \phi'$.*

Example 2.7. *Weakening is clearly sound for SAT since each assignment satisfying C_1 also satisfies $(C_1) \wedge (C_1 \vee C_2)$ and vice-versa.*

Another property which is often associated with proof systems is completeness. Intuitively, a proof system is complete if it is able to derive any logically sound information. This elegant property, also referred to as inferential completeness, is very powerful and thus usually restricted to deriving inconsistencies. This restricted form, called refutational completeness, intuitively represents the ability of a system to derive an inconsistency (usually in the form of an empty clause) from any unsatisfiable formula, which is sufficient to generate certificates of unsatisfiability for SAT, referred to as a refutations. Note that (inferential) completeness clearly entails refutational completeness while the opposite statement on any given proof system is not always true.

Definition 2.25 (Completeness)**.** *A proof system S is (inferentially) complete if, for any formulas ϕ and ϕ', $\phi \equiv \phi'$ implies $\phi \vdash_S \phi'$.*

Definition 2.26 (Refutational Completeness)**.** *A proof system S for SAT is refutationally complete if, for any unsatisfiable formula ϕ, $\phi \vdash_S \square$.*

Example 2.8. *Let S be the proof system consisting solely of the weakening rule. S is not (refutationally and inferrentially) complete for SAT since, given a variable x, it cannot derive $\square$ from $x \wedge \overline{x}$.*

Many proof systems that can be used in the context of satisfiability were introduced in the literature. One of the first and most studied systems is resolution [Rob65], which we will overview in the following sections. Other known systems include Frege [Hei13; Bus15], natural deduction [Gen64; Pra65], cutting-plane [Chv85; CCT87; Juk12], algebraic [Bea+96;

[52] The following definition is equivalent to Definition 1.17

CEI96], analytic tableau [DAg99; Häh01] and circular proof [AL19] systems. In proof theory, the relative strengths of proof systems is usually compared in terms of the size of their refutations, using the notion of simulation. We provide a common definition of simulation below [BP07a] however we may abuse the term as other variations are used in the literature [PS10; BN21b]. To give some examples, it is known that Frege and natural deduction systems are p-equivalent [CR79] and both p-simulate resolution and the cutting-plane proof systems [Goe91]. The reader can refer to the surveys in [BP01; Seg07; Bus12; BN21a] for more details on major results in proof theory.

Definition 2.27 (Polynomial Simulation)**.** *Let S and S' be two proof systems. S polynomially simulates (p-simulates) S' if there exists a polynomially-computable function f such that for any refutation π of formula ϕ in S', $f(\pi)$ is a refutation of π in S. S and S' are p-equivalent if S p-simulates S' and S' p-simulates S.*

2.3.2 Resolution Proofs for SAT

A well-known proof and refutation system for SAT is based on the resolution rule (Res) whose definition we recall below. Given two opposed clauses, this sound and complete rule for SAT deduces a resolvent clause which can be added to the formula. Resolution plays an important role in the context of SAT theory and solving. The original Davis-Putnam (DP) procedure introduced in [DP60] relies extensively on resolution to delete literals from the formula but with a worst-case exponential blow-up in its size. It was not until 1965 that resolution was established as an independent proof system in [Rob65]. Later, resolution was efficiently integrated in the context of Conflict Driven Clause Learning (CDCL) [53] thus paving the way for modern SAT solvers [MLM09]. The relationship between the resolution proof system and the CDCL framework was investigated in the literature. In [Her+08], Hertel et al. showed that clause learning can effectively p-simulate [54] resolution. This result was extended to traditional simulation, in the sense of Definition 2.27, by Pipatsrisawat and Darwiche including for CDCL augmented with the restart mechanism [PD11].

Definition 2.28 (Resolution [Rob65])**.** *Given two opposed clauses C_1 and C_2, the resolution rule is defined as follows:*

$$\frac{C_1 = x \vee A \qquad C_2 = \overline{x} \vee B}{C_3 = A \vee B}$$

Theorem 2.2 ([Rob65])**.** *Resolution is sound and refutationally complete for SAT.*

A resolution proof or derivation of a clause C from a formula ϕ is a finite sequence of resolutions starting from the clauses of ϕ and deriving C, usually represented as a finite sequence of clauses. If C is the empty clause $\square$, the proof is referred to as a (resolution) refutation of ϕ. A resolution proof can also be represented in the form of a Directed Acyclic Graph (DAG) $G = (V, A)$ whose nodes are clauses in the proof either having two or zero

[53] Refer to Section 2.2.1.2

[54] Intuitively, effective simulation corresponds to simulation modulo preprocessing, i.e., an initial encoding on the original formula.

incoming arcs, respectively when they are resolvents or clauses of the initial formula. Note that in the latter case, we refer to these clauses as leafs. We will use two typical measures to establish our results in Chapter 6: size and width. The size (or length) of a resolution derivation π, denoted $s(\pi)$ (or simply $|\pi|$), is the number of resolution steps (or resolvent clauses) in π whereas its width, denoted $w(\pi)$, is the maximum size of all its clauses.

Example 2.9. *We consider the unsatisfiable CNF formula* $\phi = (x_1) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$. *A resolution refutation* π *of* ϕ, *with* $s(\pi) = 4$ *and* $w(\pi) = 2$, *is represented in Figure 2.7.*

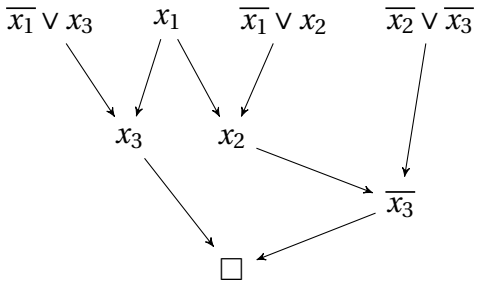


Figure 2.7: Resolution refutation as a certificate of unsatisfiability

In practice, resolution refutations can be provided as a certificate of unsatisfiability. For instance, the Booleforce solver [55] is able to generate such proofs and is also equipped with the tool TraceCheck [56] to test the validity of the generated certificate. However, memory may become a bottleneck for very large plain refutation proofs. Therefore, modern solvers usually provide clausal proofs in more adapted formats such as DRUP [57] [HJW13] or DRAT [58] [HHW13; WHH14], with more reasonable memory usage and faster proof checking which relies on Unit Propagation (UP).

2.3.3 Resolution Classes

Many restricted classes of resolution refutations have been studied in the literature namely read-once resolution [IM95], regular resolution [Tse83], tree (or tree-like) resolution refutations [Kun86] and linear resolution [Lov70] among many others[BP07b; HU09; BJ16]. In this section, we define the classes mentioned above which are necessary to establish our results in Section 6.

We start by defining read-once resolution which is a fragment encompassing resolution proofs where all the clauses are read-once, i.e., used at most once in the derivations. It was shown in [IM95] that there exists unsatisfiable CNF formulas which cannot be refuted using read-once resolution. This particular class is also of particular interest in the context of maximum satisfiability, as will be showcased in Section 3.3.2.

[55] http://fmv.jku.at/booleforce/index.html
[56] http://fmv.jku.at/tracecheck/index.html
[57] https://www.cs.utexas.edu/~marijn/drup/
[58] https://www.cs.utexas.edu/~marijn/drat-trim/

Definition 2.29 (Read-Once Resolution [IM95])**.** *A resolution proof is read-once if each clause is used at most once in the proof.*

Example 2.10. *We consider the same unsatisfiable CNF formula ϕ in Example 2.9. The resolution refutation π of ϕ represented in Figure 2.7 is not read-once since the clause $C = x_1$ is used twice in the proof.*

Next, we present another well-known class called tree (or tree-like) resolution. This class, defined below, consists of the proofs in which it is possible to have non-read-once clauses but only if they are within the original formula, having zero incoming arcs in the DAG representation. Clearly, every read-once proof is also tree-like. This particular fragment was highly investigated in the literature. In particular, tree resolution and DPLL are p-equivalent [Seg07]. Furthermore, a separation between general (or DAG) resolution and tree resolution was established in [Bon+98; BIW04], meaning that the latter cannot p-simulate the former.

Definition 2.30 (Tree Resolution [Kun86])**.** *A resolution proof is tree-like if each derived resolvent is used at most once in the proof.*

Example 2.11. *We consider the same unsatisfiable CNF formula ϕ in Example 2.9. The resolution refutation π of ϕ represented in Figure 2.7 is tree-like since the non-read-once clause $C = x_1$ is in ϕ.*

The third fragment of resolution, called regular resolution, contains proofs in which variables are resolved on at most once in each branch of the DAG, i.e., path from a clause of the initial formula to the last derived clause. A separation between general resolution and this class was also established in [Ale+02; Urq11; Vin+20]. In the context of refutations, minimal-size tree proofs are regular. This result was first stated in [Tse83] and later formally proved in [Urq95]. The proof relies on a transformation which consists in iteratively fixing irregularities through pruning. An irregularity is a sequence of clauses in the proof such that the first clause and the last one contain a literal l but at least one of the intermediate clauses does not contain this literal. The first resolution step in each irregurality is discarded and the rest of the proof is updated accordingly, potentially discarding other resolution steps which are no longer necessary, to generate a smaller-size tree-like regular refutation.

Definition 2.31 (Regular Resolution [Tse83])**.** *A resolution proof π of clause C from a CNF formula ϕ is regular if no variable is resolved twice in every path from an initial clause in ϕ to C.*

Example 2.12. *We consider the same unsatisfiable CNF formula ϕ in Example 2.9. The resolution refutation π of ϕ represented in Figure 2.7 is regular since in each branch of the DAG, all variables are resolved at most once.*

Lemma 2.1 (cf. Lemma 5.1 in [Urq95])**.** *A tree-like resolution refutation of minimal size is regular.*

Example 2.13. *We consider the formula* $\phi = (\overline{x_1}) \wedge (x_3) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee \overline{x_3})$. *The tree-like resolution refutation of* ϕ *represented on the left in Figure 2.8 is not regular since* x_1 *is resolved on two times in the same branch. As shown on the right in the same figure, this refutation can be made regular by discarding the first resolution step on variable* x_1 *and updating the rest of the proof accordingly. Notice how after the transformation, the irregularity in the branch of clause* $x_1 \vee x_2$ *is fixed and a refutation of smaller size is produced where the clauses* $\overline{x_1} \vee \overline{x_3}$ *and* x_3 *are no longer used.*

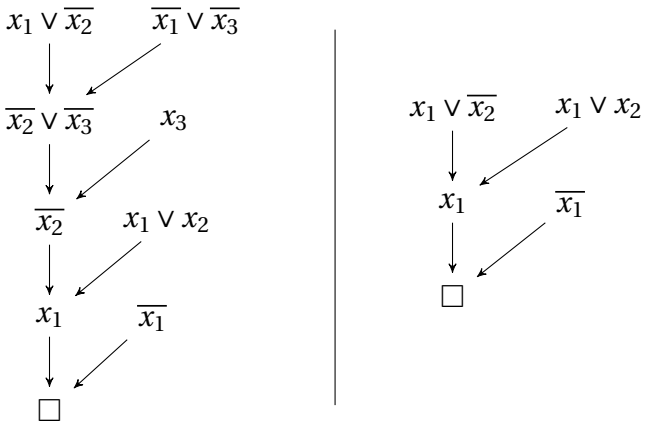


Figure 2.8: From a tree resolution refutation (left) to a regular tree resolution refutation (right)

Finally, we introduce linear resolution, formally defined below, which lies between tree-like and general resolution in terms of proof complexity [BP07b; BJ16]. In this fragment, the proofs are linear in the sense that each deduced clause is used as a premise in the next resolution step. Note that, when the first condition of (c) holds in the following definition, the clause D_i is called the input parent clause of C_{i+1}.

Definition 2.32 (Linear Resolution [Lov70])**.** *A linear resolution proof of clause* C *from a CNF formula* ϕ *is a sequence of clauses* $C_1, ..., C_m$ *such that:*

1. C_1 *is a clause in* ϕ
2. C_m *is the clause* C
3. *For every* $i < m$, C_{i+1} *is the resolvent of* C_i *either with a clause* D_i *from* ϕ *or with a clause* C_k *for some* $k < i$.

Example 2.14. *We consider the same unsatisfiable CNF formula* ϕ *in Example 2.13. The two resolution refutations of* ϕ *represented in Figure 2.7 are both linear since each deduced clause is used as premise in the next resolution step.*

2.4 Bandits for Satisfiability and Beyond

In this section, we focus on the Multi-Armed Bandit (MAB) problem which is used to study the exploration-exploitation dilemma in the context of Reinforcement Learning (RL) [SB98; WGG11; LS20]. We start by defining this well-known problem then we present the major strategies used to solve it. Furthermore, we review the use of MAB in the context of satisfiability and beyond. MAB will be used as a framework to establish our contributions in Chapter 4.

2.4.1 Multi-Armed Bandit Problem

Reinforcement Learning (RL) [SB98] is a well-known paradigm of machine learning [59] focusing on studying the way in which an autonomous agent learns from its actions. More specifically, RL investigates how intelligent agents should interact with a specific environment by exploring and exploiting certain actions, called arms or levers, while aiming to maximize the cumulative payoff. The exploration-exploitation dilemma in RL is often studied through the lens of the Multi-Armed Bandit (MAB) problem [60] [WGG11; LS20].

Formally, MAB consists of an agent and a set of candidate arms from which the agent has to choose while maximizing the expected gain. The agent relies on information in the form of rewards given to each arm and collected through a sequence of trials. MAB clearly expresses a dilemma in the tradeoff between exploitation and exploration as the agent needs to explore underused arms often enough to have a robust feedback while also exploiting good candidates which have the best rewards. The term itself is derived from gambling slot machines, which are also referred to as one-armed bandits. The first MAB model, stochastic MAB, was introduced in [LR85]. In this setting, a stochastic environment is considered where each arm is associated with an unknown probability distribution. Another well-known model is adversarial MAB, typically with a non-stationary environment where rewards can be generated without any specific probabilistic assumption on their nature [Aue+95]. Hereafter, we mainly consider the stochastic MAB framework. We start by presenting some necessary notations then we provide in Algorithm 2.4 an abstract description of the MAB framework.

Notation 2.1. *We use the following notations for the MAB framework:*

- A *denotes the set of arms where* $|A| = K \geq 2$
- T *denotes the total number of trials, referred to as horizon*
- $r_t(a)$ *denotes the reward of arm* $a \in A$ *in the trial* $t \in \{1, \ldots, T\}$
- $n_t(a)$ *denotes the number of times the arm* a *is selected during the* $t-1$ *previous runs,*

[59] Detailed reviews on other paradigms such as supervised and unsupervised learning can be found in the following textbooks [MM97; Has+09; RN20].

[60] specifically when the RL environment has a single state, in contrast to multi-state environments which are mainly stated in the form of a Markov decision processes [SB98]

- $\widehat{r}_t(a)$ *denotes the empirical mean of the rewards of arm* a *over its number of selections* $n_t(a)$ *in the* $t-1$ *previous runs, i.e.,* $\widehat{r}_t(a) = \frac{\sum_1^{t-1} r_t(a)}{n_t(a)}$.

Algorithm 2.4: Multi-Armed Bandit Framework

Input: set of arms A, horizon T

1: **for** $t = 1$ **to** T **do**
2: Agent selects arm $a \in A$ with respect to a specific strategy
3: Environment selects a reward $r_t(a)$
4: Agent observes reward
5: **end for**

Note that the strategies used by MAB to choose relevant arms are theoretically evaluated through the notion of regret [61] [LS20]. This notion, defined below, considers the difference between the expected cumulative value of the reward of the best action and its expected cumulative value for all the arms chosen by the agent through the trials. Since the objective of the agent is to find an optimal arm which maximizes the cumulative reward through the trials, each time a sub-optimal arm is chosen, the reward difference with the optimal arm is wasted. A relevant policy should thus minimize the regret throughout the trials.

Definition 2.33 (Regret [ACF02])**.** *Let* $\mu_1, \ldots, \mu_K$ *be the distributions associated with the arms in* A *in stochastic MAB. The regret* R_T *is defined as follows:*

$$R_T = \max_{a \in A} \mathbb{E}\left[\sum_{t=1}^{T} r_t(a) - \sum_{t=1}^{T} r_t(a_t)\right] = T\mu^* - \mathbb{E}\left[\sum_{t=1}^{T} r_t(a_t)\right]$$

where:

- a_t *denotes the arm chosen in the trial* $t \in \{1, \ldots, T\}$
- μ^* *is the optimal arm mean, i.e.,* $\mu^* = \max_{1 \leq i \leq K} \mu_i$

2.4.2 Strategies for MAB

In this section, we present different strategies for MAB, specifically those relevant within a stochastic context. Recall that these strategies must find a good tradeoff between exploration of underused arms and exploitation of the best ones. We can categorize the strategies into two major families: randomized and deterministic strategies. The first category contains policies which perform random exploration and, thus, which may produce different results even for a fixed setting and environment. The second category consists of strategies which are deterministic, mainly relying on smart exploration with preference to uncertainty. We start first by describing some well known randomized strategies for MAB.

[61] also called pseudo-regret in the context of stochastic MAB

ϵ-Greedy [SB98] This simple policy for stochastic MAB, which falls within the first category of randomized policies, is described in Algorithm 2.5. It relies on a probability parameter ϵ, usually of small value in $[0,1]$, to randomly explore uncharted territory. At each trial, ϵ-Greedy selects a random arm $a \in A$ with the probability ϵ, otherwise it performs greedy exploitation by selecting the arm maximizing the mean reward through the previous trials. This strategy has linear regret in T. However, in [ACF02], Auer et al. proved that ϵ-greedy can achieve logarithmic regret in T if ϵ is set as a decreasing sequence ϵ_t parametrized by $t \in \{1,...,T\}$. More specifically, if $\epsilon_t = min\{1, \frac{K}{(d^2 t)}\}$, then the regret can be bound by $O(\frac{K.ln(T)}{d^2})$ provided that $0 < d \leq min_{\mu \in \{\mu_1,...,\mu_K\}, \mu < \mu^*} \mu^* - \mu$. Finally, note that other RL policies based on the ϵ-Greedy strategy were introduced in the literature including for different variations of MAB [Tok10; Tra+10; BBG12; GSL19].

Algorithm 2.5: ϵ-Greedy

Input: set of arms A, trial number $t \in \{1,\ldots,T\}$, parameter $\epsilon \in [0,1]$
Output: an arm $a_t \in A$ to use in the trial t

1: **with** probability ϵ **do**
2: $\quad a_t \longleftarrow$ random arm $a \in A$
3: **else**
4: $\quad a_t \longleftarrow argmax_{a \in A}\, \widehat{r}_t(a)$
5: **return** a_t

Thompson Sampling (TS) [Tho33] This method was introduced in one of the earlier works on reinforcement learning and relies on the simple idea of assuming a prior distribution for every arm, mainly a beta distribution [Gup11; Eld11]. As described in Algorithm 2.6, TS maintains a beta distribution for each arm $a \in A$ defined by its two shape parameters $\alpha(a)$ and $\beta(a)$ which are initially set to 1 (lines 2-5). After each trial, the α and β values of the used arm are updated with respect to its reward (lines 7-9). TS then selects an arm maximizing the beta distribution probabilities at trial t (returned by function $Beta(\alpha,\beta)$ in line 10), which represent the likelihood of being the optimal arm. This strategy was shown to have a regret bound of $O(\sqrt{T.K.ln(K)})$ in [AG17].

Exponential-Weight Algorithm for Exploration and Exploitation (EXP3) [Aue+02] This strategy, originally introduced in the context of adversarial MAB, maintains a probability distribution $(p(a))_{a \in A}$ on the set of arms. As described in Algorithm 2.7, the probabilities are updated through weights assigned to each arm, initially set to 1 (lines 2-4) and multiplicatively updated (lines 6-7) so as to ensure a mixture of uniform and exponential probabilities in the estimated cumulative rewards of each arm (lines 8-11). EXP3 thus selects an arm according to the computed distribution in each trial (line 12) while ensuring a regret bounded by $O(\sqrt{T.K.ln(K)})$ [AB09].

Algorithm 2.6: Thompson Sampling

Input: set of arms A, trial number $t \in \{1, \ldots, T\}$
Output: an arm $a_t \in A$ to use in the trial t

if $t = 1$ **then** ▷ Initialize
  for $a \in A$ **do**
    $\alpha(a) \longleftarrow 1$
    $\beta(a) \longleftarrow 1$
  end for
else ▷ Update
  $a_{t-1} \longleftarrow$ arm selected in trial $t-1$
  $\alpha(a_{t-1}) = \alpha(a_{t-1}) + r_{t-1}(a_{t-1})$
  $\beta(a_{t-1}) = \beta(a_{t-1}) + 1 - r_{t-1}(a_{t-1})$
$a_t \longleftarrow argmax_{a \in A}\, Beta(\alpha(a), \beta(a))$ ▷ Select
return a_t

Algorithm 2.7: Exponential-Weight Algorithm for Exploration and Exploitation (EXP3)

Input: set of arms A of size K, trial number $t \in \{1, \ldots, T\}$, parameter $\gamma \in]0, 1]$
Output: an arm $a_t \in A$ to use in the trial t

if $t = 1$ **then** ▷ Initialize
  for $a \in A$ **do**
    $w(a) \longleftarrow 1$
  end for
else ▷ Update
  $a_{t-1} \longleftarrow$ arm selected in trial $t-1$
  $w(a_{t-1}) = w(a_{t-1}) * e^{\frac{\gamma}{K} * \frac{r_{t-1}(a_{t-1})}{p(a_{t-1})}}$
$s \longleftarrow \sum_{a \in A} w(a)$
for $a \in A$ **do**
  $p(a) \longleftarrow (1-\gamma)\frac{w(a)}{s} + \frac{\gamma}{K}$
end for
$a_t \longleftarrow$ random arm in A according to the distribution $\big(p(a)\big)_{a \in A}$ ▷ Select
return a_t

Next, we present a family of well-known policies of deterministic nature. One of the simplest strategies in this case is a pure exploitation strategy where no random exploration is performed. The arms can be chosen with respect to their mean reward as in the specific case of ϵ-greedy where $\epsilon = 0$. Clearly, such a naive strategy has a major flaw as it does none of the exploration required to generate relevant information for estimating the optimal arm. To circumvent this drawback, policies performing smart exploration while ensuring a certain confidence in terms of regret bounds were introduced in the literature. This family of strategies is referred to as Upper Confidence Bound (UCB) and includes two major strategies which we present hereafter.

UCB1 Strategy [ACF02] This strategy, introduced by Auer et al. in 2002 and inspired by the work of Argwal in [Agr95], is described in Algorithm 2.8. UCB1 selects the arm $a \in A$ which maximizes $UCB1(a)$ defined as follows (line 4):

$$UCB1(a) = \widehat{r}_t(a) + \sqrt{\frac{4ln(t)}{n_t(a)}}$$

where the left side term, i.e., the reward mean, aims to put emphasis on arms that received the highest rewards and, conversely, the right-side term ensures the exploration of under-used arms. UCB1 tests all the arms once (lines 1-2) to initialize the rewards and ensure that $n_t(a) \neq 0$ for all $a \in A$ before selecting the arms with respect to $UCB1(a)$. This strategy achieves a regret bound of $O(\sqrt{K.T.lnT})$. Many variations of this strategy were introduced in the literature for different MAB contexts and specifications [AMS09; Chu+11b; GC11; KCG12; Val+13; Bou+19]. One particular variation which we present next achieves optimal regret bound [AB09].

Algorithm 2.8: Upper Confidence Bound (UCB1)

Input: set of arms A, trial number $t \in \{1, \ldots, T\}$
Output: an arm $a_t \in A$ to use in the trial t

1: **if** $\exists a \in A$ s.t. $n_t(a) = 0$ **then** ▷ Initialize
2: $\quad a_t \longleftarrow a$
3: **else** ▷ Select
4: $\quad a_t \longleftarrow argmax_{a \in A}\, \widehat{r}_t(a) + \sqrt{\frac{4ln(t)}{n_t(a)}}$
5: **return** a_t

Minimax Optimal Strategy in the Stochastic Case (MOSS) [AB09] This strategy, described in Algorithm 2.9, is a variation of UCB1 which selects the arm $a \in A$ maximizing $MOSS(a)$ defined as follows:

$$MOSS(a) = \widehat{r}_t(a) + \sqrt{\frac{4}{n_t(a)} ln\left(max\left(\frac{t}{K.n_t(a)}, 1\right)\right)}$$

where the left term is similar to UCB1 while the right one, meant for exploration, additionally takes into account the number of arms and the number of executed trials. An important property of MOSS is its regret bound of $O(\sqrt{K.T})$ which is optimal for stochastic MAB [AB09] since a similar lower bound was proved by Lai and Robbins in [LR85].

Algorithm 2.9: Minimax Optimal Strategy in the Stochastic Case

Input: set of arms A, trial number $t \in \{1, \ldots, T\}$
Output: an arm $a_t \in A$ to use in the trial t
Output: an arm $a_t \in A$ to use in the trial t

1: **if** $\exists a \in A$ s.t. $n_t(a) = 0$ **then** ▷ Initialize
2: $\quad a_t \longleftarrow a$
3: **else** ▷ Select
4: $\quad a_t \longleftarrow argmax_{a \in A}\, \widehat{r}_t(a) + \sqrt{\frac{4}{n_t(a)} ln\left(max\left(\frac{t}{K.n_t(a)}, 1\right)\right)}$
5: **return** a_t

2.4.3 Applications in SAT and Beyond

Reinforcement learning, particularly in the context of MAB, has many applications in various real-life domains such as healthcare, finance and telecommunication among others[BRA20]. In recent years, there has also been a surge of interest in applying reinforcement learning techniques in the context of combinatorial problems and particularly SAT solving [YP19; SB19; Kur+19; Kur+20; Vae+20]. However, very few research studies integrate MAB-based or MAB-inspired techniques in the context of SAT. A recent example is the heuristic CHB [Lia+16a] and its variant LRB [Lia+16b], presented in Section 2.2.1.3.2, which are based on the Exoponentioal Recency Weighted Average (ERWA) [SB98]. Note that ERWA, which is used to update the variable scores in those heuristics, is used in non-stationary MAB problems to estimate the average rewards for each arm. Another more explicit application of MAB is the work of Lazaar et al. in [Laz+12] where a new approach, called Bandit Ensemble for parallel SAT Solving (BESS), was devised for parallel SAT solvers to control the clause-sharing cooperation topology, i.e., pairs of units able to exchange clauses. BESS relies on a MAB formalization of the cooperation choices and uses the UCB1 policy to pick arms at each trial.

In contrast, MAB frameworks were extensively used in the context of Constraint Programming (CP) and specifically the Constraint Satisfaction Problem (CSP) [62]. In particular, MAB frameworks were used to select the consistency level of propagation [BBP15] or a restart strategy [GS07] for CSP solving. In terms of heuristics, CHB was also implemented in the context of CSP [Sch18] and was further adapted to this problem in the form of a new heuristic, called Conflict-History Search (CHS), which similarly adopts a constraint weighting scheme on the basis of ERWA [HT21]. Rewards updated through ERWA were also used to adaptively select a backtracking strategy in [Bac+15]. Furthermore, MAB frameworks were used to select a search heuristic among a set of candidate ones at each node of the search tree in [XY18] or at each restart in [Wat+20; Kor+22]. Simple bandit-driven perturbation strategies to incorporate random choices in constraint solving with restarts were also introduced and evaluated in [PW20]. Finally, it is worth noting that upper confidence bound strategies have been used in most of these works and were shown to achieve good

[62] The reader can refer to the *Handbook of Constraint Programming* [RBW06] for a detailed overview of this problem which is closely related to SAT.

performance results.

2.5 Conclusion

In this chapter, we focused on the well-known satisfiability formalism. We defined the SAT problem and some of its known variants. Furthermore, we presented the main complete and incomplete methods to solve this problem. In particular, we introduced Conflict Driven Clause Learning (CDCL), which is a complete algorithm for SAT incorporating extensive inference during the search. We also described the major mechanisms used in modern solvers, based on CDCL, including branching heuristics and restarts among others. Then, we focused on SAT proof theory, for which we recalled the main notions. We specifically presented resolution as a proof system for SAT and we introduced some of its well-known classes. Finally, we focused on the Multi-Armed Bandit (MAB) problem, which is used to study the exploration-exploitation dilemma in the context of reinforcement learning, and we reviewed its dedicated strategies in the literature as well as its marginal use in the context of SAT. Our results in Chapter 4 revolve around SAT as we devise a MAB framework for adaptive branching by taking advantage of the restart mechanism in modern solvers. Our contributions also extend to Maximum Satisfiability (Max-SAT), the natural optimization extension of SAT which we will present in the following chapter. We particularly attempt to theoretically bridge the gap between SAT and Max-SAT inference in Chapter 6.

3 Maximum Satisfiability

Contents

In this chapter, we define the Maximum Satisfiability (Max-SAT) problem and its variants. We also review the major complete and incomplete methods for Max-SAT solving. We particularly present Branch and Bound (BnB) algorithms for Max-SAT which perform exhaustive search relying on extensive inference. We also introduce the main notions in proof theory for Max-SAT and we specifically focus on the Max-SAT resolution rule as a proof system for Max-SAT.

3.1 Definition and Variants

The Maximum Satisfiability (Max-SAT) problem is a natural optimization extension of SAT which simply consists in determining the maximum number of clauses satisfied by an assignment of a given CNF formula [1]. A more common definition in the literature is

[1] CNF formulas are represented as multisets of clauses in the context of Max-SAT

determining the optimum value of a given CNF formula ϕ defined below in the form of the minimum number of clauses that each assignment of ϕ must falsify [2]. Note that in the case where a CNF formula ϕ is satisfiable, we have $opt(\phi) = 0$, otherwise $opt(\phi) \geq 1$ since at least one clause is falsified by each assignment of ϕ. Furthermore, it is common to return an assignment which achieves the optimum value in Max-SAT, called optimal assignment. This problem is NP-hard as solving it can easily lead to a solution for the SAT problem, and therefore it does not admit a polynomial-time algorithm unless P=NP [Pap94].

Definition 3.1 (Optimum)**.** *Let ϕ be a CNF formula. The optimum of ϕ is defined as follows [3]:*

$$opt(\phi) = \min_{I \text{ assignment of } \phi} cost_\alpha(\phi)$$

.

Definition 3.2 (Maximum Satisfiability)**.** *The Maximum Satisfiability (Max-SAT) problem is defined as follows:*

> ***Max-SAT Problem***
> *Input: a CNF formula ϕ*
> *Output: $opt(\phi)$*

Theorem 3.1 ([Pap94])**.** *The Max-SAT problem is NP-hard.*

Example 3.1. *We consider the formula $\phi = (x_1) \wedge (x_2) \wedge (\overline{x_2}) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$ and the assignments $\alpha = \{\overline{x_1}, x_2, x_3\}$ and $\alpha^* = \{x_1, x_2, x_3\}$. We have $cost_\alpha(\phi) = 3 > cost_{\alpha^*}(\phi) = 2$. Since no other assignment of ϕ achieves a smaller cost, α^* is an optimal assignment and we have $opt(\phi) = 2$.*

A simple variant of Max-SAT is Max-k-SAT which is a natural optimization extension of k-SAT, taking a k-CNF formula as input [4]. Moreover, there are many extensions of Max-SAT each providing a more generic abstraction to deal with broader constraints. In the following definitions, we present a first variant called Partial Max-SAT in which partial formulas are considered. Such formulas contain a set of hard clauses that must be satisfied and a set of soft clauses to be optimized similarly to plain Max-SAT. Clearly, every plain Max-SAT instance can be represented as a partial Max-SAT formula where all clauses are soft.

Definition 3.3 (Partial Formula)**.** *A partial CNF formula is a bipartite set of clauses $\phi = H \cup S$ where H is the set of hard clauses that must be satisfied and S is the set of soft clauses as in plain Max-SAT.*

Definition 3.4 (Partial Max-SAT)**.** *The Partial Maximum Satisfiability (Partial Max-SAT) problem is a variant of Max-SAT defined as follows:*

> ***Partial Max-SAT Problem***
> *Input: Partial CNF formula $\phi = H \cup S$*
> *Output: $opt(S)$ s.t. all the clauses in H are satisfied, "no solution" otherwise*

[2] in which case returning the value $|\phi| - opt(\phi)$ corresponds to an equivalent output of the original definition
[3] The cost of an assignment $cost_\alpha(\phi)$ corresponds the number of clauses in ϕ falsified by α, as specified in Definition 2.11.
[4] refer to Section 2.1

Another variant of Max-SAT is the Weighted Maximum Satisfiability (Weighted Max-SAT) problem, defined below, where positive finite weights are associated with each clause in the formula to represent the penalty of falsifying it. Weighted Max-SAT consists in determining the maximal sum of weights of clauses falsified by an assignment of the variables. Note that this problem can be combined with the partial version defined above by authorizing infinite weights. This most generic variant, called Partial Weighted Max-SAT, considers weighted formulas where hard clauses have infinite weight and must be satisfied, otherwise the returned optimum value would also be infinite. Note that in the following sections we mainly consider the plain Max-SAT problem but we may provide further comments on its variants when necessary [5].

Definition 3.5 (Weighted Formula)**.** *A weighted CNF formula is a set of tuples* (C, w) *where* C *is a clause and* $w \in N^*$ *is its associated weight, denoted* $w(C)$*.*

Definition 3.6 (Weighted Cost)**.** *Let* ϕ *be a weighted CNF formula and* α *be an assignment of* ϕ*. The (weighted) cost of* α *is defined as follows:*

$$cost_\alpha(\phi) = \sum_{(C,w)\in\phi \text{ s.t. } C|_\alpha=\square} w$$

Definition 3.7 (Weighted Max-SAT)**.** *The Weighted Maximum Satisfiability (Weighted Max-SAT) problem is a variant of Max-SAT defined as follows:*

> ***Partial Max-SAT Problem***
> *Input: a weighted CNF formula* ϕ
> *Output:* $opt(\phi)$

It is important to note that Max-SAT soundness notions on formula transformations differ from SAT [ABL13]. In particular, Max-SAT equivalence preservers the cost of each assignment as defined below. Another notion, most commonly used in the context of Max-SAT solving, is Max-SAT reducibility which allows to performs sound encodings as long as the original cost of each assignment is maintained. A third weaker notion is equioptimality which simply preserves the optimum value and can be seen as the extension of equisatisfiability [6] for Max-SAT. Hereafter, we provide a formal definition of these three notions.

Definition 3.8 (Equivalence)**.** *Let* ϕ *and* ϕ' *be two CNF formulas.* ϕ *is equivalent (in the sense of Max-SAT) to* ϕ'*, denoted* $\phi \equiv \phi'$*, if for any assignment* $\alpha : var(\phi) \cup var(\phi') \longrightarrow \{True, False\}$*, we have* $cost_\alpha(\phi) = cost_\alpha(\phi')$*.*

Definition 3.9 (Reducibility)**.** *Let* ϕ *and* ϕ' *be two CNF formulas.* ϕ *is Max-SAT reducible to* ϕ' *if for any assignment* $\alpha : var(\phi) \longrightarrow \{True, False\}$*, we have* $cost_\alpha(\phi) = cost_\alpha(\phi')$*.*

Definition 3.10 (Equioptimality)**.** *Let* ϕ *and* ϕ' *be two CNF formulas.* ϕ *is equioptimal to* ϕ' *if* $opt(\phi) = opt(\phi')$*.*

[5] In particular, we may abuse the term weighted Max-SAT to refer to both weighted variants with or without infinite weights.

[6] refer to Definition 2.10

The Max-SAT problem and its variants are powerful formalisms which can be used to represent and solve many real-world and crafted problems making it of great academic and industrial interest [7]. In particular, Max-SAT can be used to encode and solve several well-known academic problems such as Minimum Satisfiability (Min-SAT) [Li+10b; Küg12; Zhu+12], Maximum Clique (Max-CLIQUE) [Cha+97; LQ10; FLX16; Jia+18] and Maximal CSP (Max-CSP) [Arg+08; Arg+09a; Arg+09b] among others. Major industrial applications of Max-SAT and its variants include scheduling [AN14; Bof+15; DM17; DMW19; Lia+19; Lia+21], explainable AI [HSJ17; Ign+18; MM18; GM19; CSJ20], security [WQL09; BDS11; LZK16; Fen+17; SA20], hardware and software debugging [Che+09; Che+10; MVN12; Saf+07; JM11], routing [JKS95; XRS03; FM06a; Li+20c; GLM14], data analysis [BJ13; Hyt+17; BHJ18] and bioinformatics [Gra+10; GL12; GML11; JCZ16]. The expressiveness of Max-SAT, its close association with SAT and its various applications explain the importance of studying this problem in terms of theory and solving. This importance is also attested by the Max-SAT evaluation events [8] which are organized each year since 2006 to evaluate current state-of-the-art Max-SAT solvers and to incite further improvement in their empirical performance.

3.2 Algorithms

In this section, we present the major complete algorithms for solving the Max-SAT problem. We first review Branch and Bound (BnB) algorithms for Max-SAT which explore the search space through an exhaustive branching over the variables in the formula. Then, we focus on SAT-based approaches which take advantage of the power of SAT solvers as decision engines. We also briefly present other complete methods for Max-SAT including direct encodings to other optimization problems as well as sequential portfolio algorithms. Finally, to be exhaustive, we briefly review incomplete methods for Max-SAT.

3.2.1 Branch and Bound

Branch and Bound (BnB) is one of the oldest methods introduced for solving optimization problems in the literature [LD60]. Its name was first suggested by Little et al. in [Lit+63] were it was used to solve the Traveling Salesman Problem (TSP) [9]. Later, BnB became a common tool for solving various optimization problems including Max-SAT [BT83; CZ12; LM21]. In the subsequent sections, we first start by presenting the outline of a BnB algorithm for Max-SAT. We present the inference mechanisms used in such algorithms. Then, we focus on the lower bound estimation component and the different learning schemes used in the literature. Finally, we recall a recently introduced property aiming to explain the results achieved by BnB Max-SAT solvers in the last decade and which will be used to establish our contributions in Chapter 5.

[7] A detailed overview of Max-SAT applications can be found in [BJM21].
[8] https://maxsat-evaluations.github.io/
[9] refer to Example 1.4

3.2.1.1 Outline

A Branch and Bound (BnB) algorithm for Max-SAT explores the search space by constructing a search tree while maintaining two values: the Upper Bound (UB) and the Lower Bound (LB) which correspond respectively to the best computed solution so far and to the estimation of the best accessible solution. In each node of the search tree, the current assignment is extended, if possible, using different inference rules. Then, the value of LB is computed and compared to UB. If it is greater, finding a better solution in the current subtree is not possible and, thus, a cut is performed by forcing the algorithm to backtrack. A simple estimation of LB in the current node is the number of falsified clauses by the current assignment. If a complete assignment is reached, the UB value and the optimal assignment are updated and a backtrack is performed. Otherwise, an unassigned variable is chosen using a specific branching heuristic. These steps are repeated until the whole search space is explored. As the efficiency of a BnB algorithm depends on the quality of UB and LB, BnB solvers rely on a variety of techniques and heuristics mainly dedicated to generate more cuts in the search tree. Major BnB solvers in the literature include MiniMaxSat [HLO07; HLO08], MaxSatz [LMP07; Li+08; Li+09; Li+10a], Akmaxsat [Küe12], Ahmaxsat [AH15a; Abr15] and the recent MaxCDCL solver [Li+21a; Li+22a]. The outline of a BnB algorithm for Max-SAT is presented in Algorithm 3.1 where:

- UB contains the current upper bound value which is updated each time a complete assignment is visited. When the search space has been fully explored, UB would contain the optimum value of the input formula which is to be returned by the algorithm. Note that an initial value for UB can be set by considering the number of clauses in the input formula or by computing the cost of a random assignment. Another more commonly used method is to launch a local search algorithm dedicated to Max-SAT [10], to rapidly generate a good initial UB value, as suggested first in [BF98].
- α represents the current assignment, updated when branching, backtracking or when a semantic inference rule, used to assign further literals, is applied.
- LB contains the estimation of the lower bound in the current node which is computed through the function **Compute_LB**(ϕ, α). In its simplest form, this function returns the number of clauses in ϕ falsified by the current assignment α. However better LB estimations can be computed as will be explained in Section 3.2.1.3. A good LB estimation helps to prune the search tree by performing cuts whenever $LB \geq UB$.
- **Inference_Apply**(ϕ, α) denotes the application of syntactic or semantic rules with respect to the current formula, i.e., $\phi|_\alpha$, which is updated alongside the current assignment when necessary.
- **Backtrack**(ϕ, α) undoes all the assignments made within and after the backtrack level β by doing the necessary updates on ϕ and α.
- **Branching_Heuristic**(ϕ, α) picks a promising literal to branch on in the search tree.

[10] refer to Section 3.2.4

Algorithm 3.1: Branch and Bound (BnB) for Max-SAT

Input: a CNF formula ϕ
Output: $opt(\phi)$

```
1:  UB ⟵ |φ|                                          ▷ Upper Bound
2:  α ← ∅                                             ▷ Assignment
3:  repeat
4:      Apply_Inference(φ, α)                         ▷ Inference
5:      LB ⟵ Compute_LB(φ, α)                         ▷ Estimation
6:      if LB ≥ UB then                               ▷ Cut
7:          Backtrack(φ, α)
8:      else
9:          if |I| = |var(φ)| then
10:             UB ⟵ LB
11:             Backtrack(φ, α)                       ▷ Backtracking
12:         else
13:             l ⟵ Branching_Heuristic(φ, α)         ▷ Branching
14:             α ⟵ α ∪ {l}
15: until |α| = 0
16: return UB
```

Example 3.2. *We consider the formula* $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$ *where* $C_1 = x_1$, $C_2 = x_2$, $C_3 = \overline{x_2}$, $C_4 = \overline{x_1} \vee x_3$, $C_5 = \overline{x_1} \vee x_2$ *and* $C_6 = \overline{x_2} \vee \overline{x_3}$. *A search tree depicting the execution of the BnB algorithm on* ϕ *with lexicographic branching (i.e.,* $x_1 < x_2 < x_3$*) is showcased in Figure 2.1. The initial upper bound value is* $UB = |\phi| = 6$. *Assigning literal* x_1 *does not falsify any clause in* ϕ *and therefore we have* $LB = 0$. *Next, the algorithms assigns the literals* x_2 *and* x_3 *leading to the falsification of the clauses* C_3 *and* C_6 *and thus to the respective LB values 1 and 2. Since all the variables are assigned, the UB value is updated and becomes 2. When assigning* $\overline{x_3}$ *we have* $LB = 2 \geq UB$ *and thus a cut is performed. The algorithm continues to explore the search space while performing cuts when necessary until all possible assignments have been visited or discarded. Finally, the last* UB *value corresponds to the optimum of* ϕ*, i.e.,* $opt(\phi) = 2$*, achieved by the assignment* $\alpha = \{x_1, x_2, x_3\}$.

Complete solvers for SAT and BnB solvers for Max-SAT share many aspects. Indeed, practically all branching heuristics for (weighted) Max-SAT BnB are similar, based on, or inspired by those devised for SAT. In particular, the first solvers dedicated for Max-SAT used the traditional Jeroslow-Wang (JW) [JW90] and MOMS [ZM88; Fre95] heuristics [11], sometimes with very small variations [BF98; AMP03; Pla03; XZ05; AMP05; LMP05]. More recent solvers introduced more variations to further adapt these heuristics to (weighted) Max-SAT. The aim is to focus on generating shorter clauses and specifically unit clauses, which enables a better estimation of the lower bound, and also to balance the search by considering opposed literal polarities for each variable. In particular, MaxSatz and Ahmaxsat use a heuristic based on MOMS, taking into account the presence of literals in unit, binary and

[11] refer to Section 2.2.1.3.2

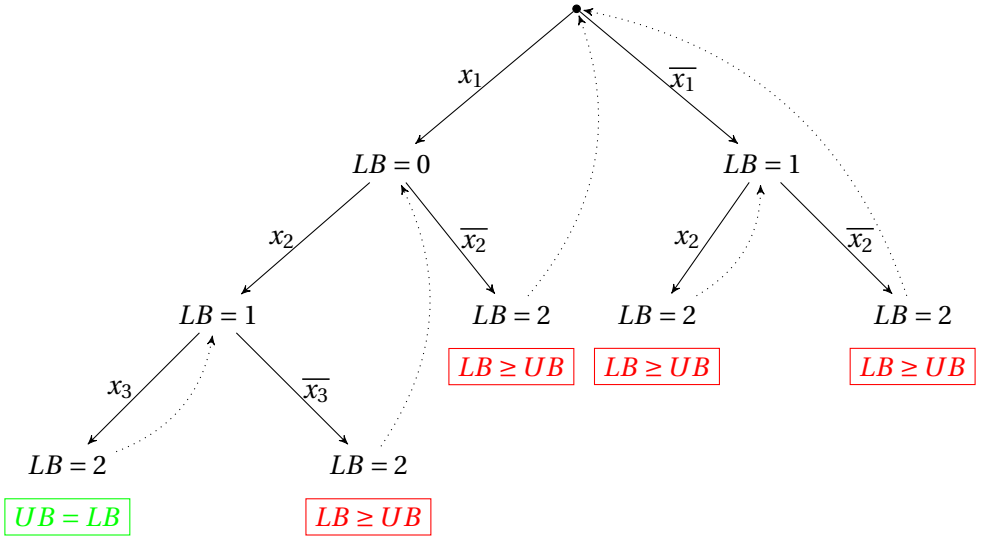


Figure 3.1: Search tree depicting the execution of a BnB algorithm for Max-SAT where branching is made in lexicographic order of the variables. Solid and dotted arrows represent respectively branching and backtracking.

ternary [12] clauses. Akamaxsat [LMP07; Li+10a] uses a similar heuristic while additionally considering the presence of literals within inconsistent subsets of the formula during lower bound estimation, which will be explained in Section 3.2.1.3. MiniMaxSat [HLO07; HLO08] uses a hybrid heuristic scheme in the context of weighted partial Max-SAT where VSIDS is used only on hard clauses and a weighted version of Jerslow-Wang is used otherwise. MaxCDCL [Li+21a; Li+22a] is also based on a hybrid scheme, similar to the one used in the MapleCOMSPS SAT solver [Lia+16c], which alternates alternates LRB phases and VSIDS phases while using Luby restarts for the former and glucose-style restarts for the for latter. Other SAT mechanisms incorporated into Max-SAT BnB include lazy data structures first used in [AMP04; AMP05; AM06] and later in MiniMaxSat and MaxCDCL. SAT inference rules which are sound for Max-SAT can also be used in BnB as will be showcased in the following section.

3.2.1.2 Inference

In each node of the search tree, BnB algorithms use a variety of inference rules to simplify the formula, extend the current assignment or efficiently compute a better lower bound estimation. Syntactic rules apply a sound transformation on the formula, usually by maintaining Max-SAT equivalence [13], and are used for simplification to reduce the size of the clauses or the formula. Semantic inference rules enable to assign further literals while maintaining equioptimality and thus help to prune the search space. It is important to note that rules applicable for SAT are not necessarily sound for Max-SAT. For instance, resolution [DP60; Rob65] and Unit Propagation (UP) [DP60; DLL62] are not valid

[12] clauses that contain exactly three literals
[13] refer to Section 3 for soundness notions

for Max-SAT [BLM07; LMP05]. Hereafter, we start by presenting valid semantic rules for Max-SAT.

Pure Literals (PL) [DP60; DLL62] This rule originally used in the context of SAT solving [14] is also valid for Max-SAT. Indeed, assigning a literal with a unique polarity can be used to extend the current assignment as it maintains equioptimality [NR00].

Hard Unit Propagation (HUP) [BF98] This rule consists in assigning literal l when $LB = UB - 1$ and a unit clause $C = \overline{l}$ is in the current formula. Indeed, assigning $\overline{l}$ cannot lead to a better assignment as it already achieves a previously established UB value. In the weighted case, this rule is applied if we have:

$$\sum_{C=\overline{l}} w(C) \geq UB - LB$$

.

Dominant Unit Clause (DUC) [NR00] This rule consists in assigning literal l if the number of unit clauses containing l is greater or equal to the number of clauses where $\overline{l}$ appears. The sum of weights of the clauses can be used in the context of weighted Max-SAT.

In terms of Syntactic inference rules, BnB algorithms for Max-SAT mainly rely on the Max-SAT resolution rule [LH05a; BLM06; BLM07; LHG08], defined below. This sound and complete rule for Max-SAT [15] can be considered as extension of the resolution rule used in the context of SAT [Rob65]. One major difference between resolution and Max-SAT resolution is that the former adds the conclusions without deleting the premises whereas the latter replaces the premises in by the conclusions. Furthermore, Max-SAT resolution produces many clauses including a resolvent clause and a set of compensation clauses. As their name suggests, these clauses are necessary to preserve equivalence in the sense of Max-SAT. Note that tautological clauses generated by Max-SAT resolution can be discarded.

Definition 3.11 (Max-SAT resolution [LH05a; BLM06; BLM07; LHG08])**.** *Given two clauses $C_1 = x \vee A$ and $C_2 = \overline{x} \vee B$ s.t. $A = a_1 \vee ... \vee a_s$ and $B = b_1 \vee ... \vee b_t$, the Max-SAT resolution rule is defined as follows:*

[14] refer to Section 2.2.1.1

[15] We give a more detailed overview of Max-SAT resolution as a proof system in Section 3.3.

$$\frac{C_1 = x \vee A \qquad C_2 = \overline{x} \vee B}{\begin{array}{c} Cr = A \vee B \\ CC_1 = x \vee A \vee \overline{b_1} \\ CC_2 = x \vee A \vee b_1 \vee \overline{b_2} \\ \vdots \\ CC_t = x \vee A \vee b_1 \vee ... \vee b_{t-1} \vee \overline{b_t} \\ CC_{t+1} = \overline{x} \vee B \vee \overline{a_1} \\ CC_{t+2} = \overline{x} \vee B \vee a_1 \vee \overline{a_2} \\ \vdots \\ CC_{t+s} = \overline{x} \vee B \vee a_1 \vee ... \vee a_{s-1} \vee \overline{a_s} \end{array}}$$

where Cr is the resolvent clause and $CC_1, ..., CC_{t+s}$ are compensation clauses.

Example 3.3. *The application of the Max-SAT resolution rule on clauses $C_1 = x_1 \vee x_2$ and $C_2 = \overline{x_1} \vee \overline{x_3}$ generates the resolvent clause $Cr = x_2 \vee \overline{x_3}$ and the compensation clauses $CC_1 = x_1 \vee x_3$ and $CC_2 = \overline{x_1} \vee \overline{x_1}$*

In practice, specific cases of Max-SAT resolution or hyper Max-SAT resolution patterns, i.e., patterns involving a sequence of Max-SAT resolution applications, are used to simplify the formula or to detect and transform inconsistent subsets of the formula in order to enhance the lower bound estimation process as will be explained in the following section. Hereafter we present some of the generic syntactic rules used by BnB Max-SAT solvers in the literature.

Almost Common Clauses (ACC) [BR99] This rule also known as symmetric cut or neighborhood resolution [LH05a] is a specific case of Max-SAT resolution, formally defined below, where A is equal to B in Definition 3.11.

Definition 3.12 (Symmetric cut [BR99])**.** *Given two clauses $C_1 = x \vee A$ and $C_2 = \overline{x} \vee A$ where A is a disjunction of literals, the symmetric cut rule is defined as follows:*

$$\frac{C_1 = x \vee A \quad C_2 = \overline{x} \vee A}{A}$$

Chain Resolution [LMP07; LHG08] This rule is a hyper resolution pattern deducing the empty clause without increasing the size of the clauses and the formula. It is mainly used in the context of lower bound estimation. Given $k \geq 1$, it has the following form:

$$\frac{l_1,\ \overline{l_1} \vee l_2,\ \overline{l_2} \vee l_3,\ \ldots,\ \overline{l_{k-1}} \vee l_k,\ \overline{l_k}}{\square,\ l_1 \vee \overline{l_2},\ l_2 \vee \overline{l_3},\ \ldots,\ l_{k-1} \vee \overline{l_k}}$$

Cycle Resolution [LHG08] This rule is also a hyper resolution pattern used in lower bound estimation which, given $k \geq 2$, has the following form:

$$\frac{\overline{l_1} \vee l_2,\ \overline{l_2} \vee l_3,\ \ldots,\ \overline{l_{k-1}} \vee \overline{l_k},\ \overline{l_k} \vee \overline{l_1}}{\overline{l_1},\ (l_1 \vee \overline{l_i} \vee l_{i+1})_{2 \leq i < k},\ (\overline{l_1} \vee l_i \vee \overline{l_{i+1}})_{2 \leq i < k}}$$

Note that in the context of weighted Max-SAT, the weighted version of the Max-SAT resolution introduced in [BLM07; LHG08] can be used [16]. In general, the above patterns remain similar as we can suppose without loss of generality that the clauses have similar weight. Indeed, in the context of weighted Max-SAT, the fold and unfold rules are used to deal with weights. In particular, the unfold rule enables to partition clauses with respect to the minimum weight and thus to produce copies such that the patterns above, containing clauses with similar weights, can be identified. In contrast, the fold rule enables to merge similar clauses when necessary in order to simplify the formula.

Definition 3.13 (Fold & Unfold)**.** *Given a clause C and two positive weights w_1 and w_2, the fold and unfold rules are respectively defined as follows:*

$$\frac{(C, w_1) \quad (C, w_2)}{(C, w_1 + w_2)} \qquad \frac{(C, w_1 + w_2)}{(C, w_1) \quad (C, w_2)}$$

Other forms of inference that can be incorporated in BnB for Max-SAT include preprocessing mechanisms [BMM13; BSJ15a; BSJ15b; Kor+17; IBJ22] and SAT-inspired clause-learning techniques [17]. [DCB10b; AH16; Li+21a; Li+22a]. While the former can be applied on the formula in any solving context prior to the search, the latter specifically pertains to Max-SAT BnB and recent works mainly focus on learning clauses when cuts are performed in the search tree. Such a mechanism was first used in [AH16] where a hard clause, referred to as nobetter clause, is learned to forbid a partial assignment which cannot lead to a better solution. To this end, the authors extend the traditional implication graph notion in SAT [SS96] to encompass reasons of the semantic inference rules presented above, such that an analysis on the graph would generate a clause capturing the reasons of the cut. However, the computed nobetter clauses are usually of large size. In [Li+21a; Li+22a], Li et al. circumvent this problem by renaming clauses using newly introduced soft literals. More specifically, each soft clause C becomes hard after adding the hard encoding of $l \leftrightarrow C$ in CNF form to the formula where l is a new soft literal. The authors thus devise a new solver on top of an existing SAT solver where clauses are learned both when a hard clause is falsified as in CDCL SAT solvers, similarly to the earlier attempt in [DCB10b], or when a soft conflict occurs, i.e., when the current partial assignment cannot be extended to a complete one falsifying fewer than UB soft clauses. The authors show that such a mechanism can make Max-SAT BnB competitive on industrial instances compared to the SAT based approaches presented in Section 3.2.2.

3.2.1.3 Lower Bound Estimation

The LB estimation is one of the most critical components of a BnB Max-SAT algorithm. Indeed, computing lower bounds with better quality entails more cuts in the search tree

[16] refer to Section 3.3.3

[17] clause-learning techniques using rules described above during lower bound estimation will be accounted for in the following sections

and, thus, faster solving. However, as these computations are done frequently during the search, the time devoted to them is closely correlated to the efficiency of the BnB algorithm. Therefore, it is important to strike a balance between the computation time for LB and its quality. A simple estimation for LB is the number of falsified clauses in the formula ϕ by the current assignment α, i.e., $LB = cost_\alpha(\phi)$. As this simple estimation is often quite far from the best accessible solution, BnB Max-SAT solvers refine it by calculating the number of disjoint Inconsistent Subsets (ISs), also referred to as cores, in the current formula $\phi|_\alpha$.

Definition 3.14 (Inconsistent Subset)**.** *Let ϕ be an unsatisfiable CNF formula. An Inconsistent Subset (IS) (or core) of ϕ is an unsatisfiable set of clauses $\Psi \subseteq \phi$.*

To detect ISs, BnB algorithms use Simulated Unit Propagation (SUP) [LMP05]. SUP replicates the Unit Propagation (UP) mechanism used in SAT solvers which, as described in Section 2.2.1.1, consists in iteratively satisfying the literals appearing in unit clauses until a conflict is found, i.e., an empty clause is generated. Note that UP is not a valid inference rule for Max-SAT, i.e., it does not necessarily maintain equioptimality [LMP05], and therefore, the variables assigned by UP are not added to the current assignment after LB estimation. More specifically, these variable assignments are stored in a separate temporary trail which is discarded after the LB computation process. That is why we say that BnB Max-SAT algorithms simulate UP, instead of actually applying it, and we refer to this mechanism as simulated UP.

The propagation steps generated by SUP can be formally represented in the form of an implication graph, defined below in the context of BnB solvers for Max-SAT. It is important to note that implication graphs used in the context of LB estimation in Max-SAT BnB are different from the traditional ones used in CDCL SAT [18]. In particular, the graph is only representative of the propagation steps performed during SUP in the last decision level. As such, the graph does not include decision nodes. Nodes with zero incoming arcs represent the literals present in unit clauses which form the first propagation layer in the SUP mechanism. In [AH14c], Abramé and Habet propose to maintain all SUP propagation reasons in the implication graph by considering all the unit clauses causing the assignment of the variables and show that this can help efficiently handle the SUP process as it enables to undo propagations in non-chronological order thus reducing the number of redundant propagation steps made in BnB solvers.

Definition 3.15 (Implication Graph of an IS)**.** *Let ψ be an IS of a CNF formula ϕ and α an assignment. We suppose that exactly one clause is falsified by α (SUP stopped when the first empty clause is generated). An implication graph of ψ is a directed acyclic graph $G = (V, A)$ defined as follows:*

- $V = \{l \in \alpha\} \cup \{\diamond_C \mid C \in \psi \text{ and } |C| = 1\} \cup \{\square\}$
- $A = \{(l, l', C) \mid l, l' \in \alpha \text{ and } C \in \psi \text{ is reduced by } l \text{ and propagates } l'\} \cup$
  $\{(\diamond_C, l, C) \mid l \in \alpha \text{ and } C = \{l\} \in \psi\} \cup$
  $\{(l, \square, C) \mid l \in \alpha \text{ and } C \in \psi \text{ is falsified by } \alpha \text{ and } \bar{l} \in \psi\}$

[18] refer to Section 2.2.1.2

The directed edges are labeled by clauses and the nodes $\diamond$ are omitted in G.

Notation 3.1. *For an implication graph G and a literal $l \in V$ (possibly $\square$), we denote :*

- $pred_G(l)$ *(resp.* $succ_G(l)$*) the predecessors (resp. successors) of* l *in* G
- $neigh_G(l) = pred_G(l) \cup succ_G(l)$
- $src_G(l)$ *the clause that lead to the propagation of the literal* l *in* G

The index G will be omitted if there is no confusion.

Example 3.4. *We consider the IS $\psi = \{x_1, x_2, x_3, \overline{x_3} \vee \overline{x_4}, \overline{x_1} \vee \overline{x_2} \vee x_4\}$. An implication graph corresponding to a propagation sequence leading to the detection of ψ is represented in Figure 3.2.*

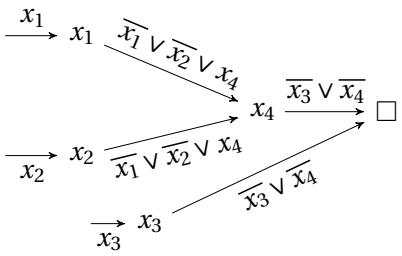


Figure 3.2: Implication graph representing the SUP steps in Max-SAT BnB

When an IS is detected, it must be counted only once in the LB estimation. To this end, detected ISs are treated using two methods to ensure that they are disjoint. They are either temporarily deleted or transformed by Max-SAT resolution. The deletion of ISs has several advantages: it is less time-consuming and it doesn't increase the size of the formula. However, it produces a formula that is not equivalent to the original and, thus, that may contain less ISs. Therefore, this method applies local changes on the formula that are only preserved during the LB estimation in the current node.

The second method to treat ISs, outlined in Algorithm 3.2, relies on a local application of the Max-SAT resolution rule [19] [AH14b]. A Max-SAT resolution transformation requires an implication graph describing the sequence of propagations leading to the detection of the IS. The transformation is usually done in the reverse order of propagation. The treatment performed by Apply_Resolution($\phi, C_1, C_2, var(l)$) (line 6) consists in deleting the clauses C_1 and C_2 from the formula and adding the resolvent (returned in line 6) and compensation clauses described in the Max-SAT resolution rule.

Definition 3.16 (Transformation of an IS)**.** *Let ψ be an IS and $S = \langle x_1, ..., x_k \rangle$ be a sequence of variables appearing in ψ. The Max-SAT resolution transformation of ψ with respect to S, denoted $\Theta(\psi, S)$, is the set of clauses obtained from ψ after the application of Max-SAT*

[19] refer to Definition 3.11

resolution steps in accordance to the sequence S, i.e., $\Theta(\psi, S) = \theta(\theta...(\theta(\psi, x_1), x_2)..., x_k)$ *where* $\theta(\psi, x)$ *denotes the application of the Max-SAT resolution step on two clauses* C *and* C' *s.t.* $x \in C$ *and* $\overline{x} \in C'$.

Remark 3.1. *A transformation* $\Theta(\psi, S)$ *is not unique in general as the application of Max-SAT resolution steps with respect to a given variable sequence S is not deterministic and, thus,* $\Theta(\psi, S)$ *represents any of the possible outcomes after the transformation of* ψ *with respect to S. However, if S represents the reverse propagation order, the transformation with respect to S becomes deterministic and* $\Theta(\psi, S)$ *becomes unique.*

Algorithm 3.2: Max-SAT resolution transformation of an IS in Max-SAT BnB

Input: an IS ψ, an implication graph G of ψ
Output: $\Theta(\psi, S)$ where S represent the reverse propagation order sequence

1: $C_1 \leftarrow src_G(\square)$
2: **while** $C_1 \neq \square$ **do**
3: $\quad l \leftarrow$ the last propagated literal in G
4: $\quad C_2 \leftarrow src_G(l)$
5: $\quad C \leftarrow$ Apply_resolution$(\psi, C_1, C_2, var(l))$
6: $\quad C_1 \leftarrow C$
7: **return** ψ

Example 3.5. *We consider the same IS* $\psi = \{x_1, x_2, x_3, \overline{x_3} \vee \overline{x_4}, \overline{x_1} \vee \overline{x_2} \vee x_4\}$ *in Example 3.4 detected through the implication graph represented in Figure 3.2. The Max-SAT resolution transformation of* ψ *with respect to the variable sequence* $S = \langle x_4, x_3, x_2, x_1 \rangle$ *(reverse order of propagation) is given in Figure 3.3. After the transformation, we obtain* $\Theta(\psi, S) = \{\square, \overline{x_1} \vee \overline{x_2} \vee x_3 \vee x_4, x_1 \vee \overline{x_3} \vee \overline{x_4}, \overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4}, x_1 \vee x_3, \overline{x_1} \vee x_2 \vee x_3, x_1 \vee x_2\}$.

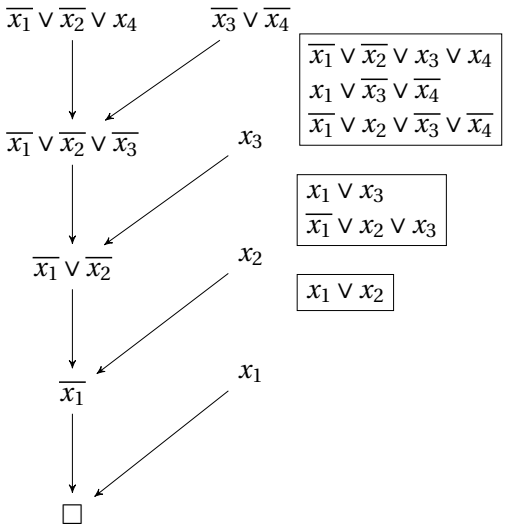


Figure 3.3: Max-SAT resolution transformation of an IS in Max-SAT BnB. Compensation clauses for each step are represented in boxes.

It is important to note that, unlike the deletion method, the changes entailed by Max-SAT transformations can be maintained in the sub-tree thus ensuring that the detected IS is computed and counted only once. This is possible since Max-SAT resolution produces an equivalent formula and this entails an incremental calculation of LB while reducing redundancy in the detection of ISs. Nevertheless, this method has several shortcomings: it is time-consuming and it may increase the size of the formula by adding the compensation clauses obtained after each Max-SAT resolution step. We explain in the next section how state of the art solvers overcome these limits.

3.2.1.4 Learning Schemes

One of the major challenges for Max-SAT BnB solvers is to perform efficient Max-SAT resolution transformations of ISs. There are different learning schemes in the literature. The first scheme implemented in the MiniMaxSat solver learns a transformation only if all the intermediary resolvents contain less than four literals[HLO08]. A family of other schemes consists in learning transformations matching particular patterns. These patterns are mainly described by inference rules that can be deduced from Max-SAT resolution. They can be specific cases of Max-SAT resolution or a combination of several Max-SAT resolution steps as described in Section 3.2.1.2. We give below three major patterns used in state-of-the-art solvers:

$$\frac{l_1 \vee l_2,\ l_1 \vee \overline{l_2}}{l_1}\ (P_1)$$

$$\frac{l_1 \vee l_2,\ l_1 \vee l_3,\ \overline{l_2} \vee \overline{l_3}}{l_1,\ l_1 \vee l_2 \vee l_3,\ \overline{l_1} \vee \overline{l_2} \vee \overline{l_3}}\ (P_2)$$

$$\frac{l_1,\ \overline{l_1} \vee l_2,\ \overline{l_2} \vee l_3,\ \ldots,\ \overline{l_{k-1}} \vee l_k,\ \overline{l_k}}{\square,\ l_1 \vee \overline{l_2},\ l_2 \vee \overline{l_3},\ \ldots,\ l_{k-1} \vee \overline{l_k}}\ (P_3)$$

P_1 a particular case of the ACC rule (where $A = l_1$ in Definition 3.12) [BR99]. Pattern P_2 is a specific case of cycle resolution (where $k = 2$) while P_3 corresponds to the chain resolution rule [LMP07; LHG08]. The particular case of P_3 where $k = 1$ is also referred to as the complementary unit clause rule in the literature, which can be used to replace two opposed unit clauses with an empty clause [NR00] (also a particular case of ACC where $A = \emptyset$ in Definition 3.12). A pattern can cover an IS entirely as in pattern P_3, or partially as in patterns P_1 and P_2. These patterns present several advantages: they can be easily identified and they don't increase the number of clauses in the formula. Furthermore, patterns P_1 and P_2 produce a unit resolvent clause which enables the solver to detect more ISs using SUP. More generic patterns, called Unit Clause Subsets (UCS), were introduced and empirically studied in [AH14d]. Hereafter, we give the formal definition of these patterns and we explain how they can be easily detected using the implication graph of an IS.

Definition 3.17 (Unit Clause Subset [AH14d])**.** *Let ϕ be a CNF formula and $k \geq 2$ be a natural number. A k-Unit Clause Subset, denoted k-UCS, is a set of clauses $\{C_1, ..., C_k\} \subseteq \phi$ such that there exists a sequence of Max-SAT resolution steps on $C_1, ..., C_k$ that produces a*

unit clause reslovent. In particular, if $\forall i \in \{1,...,k\}$ *we have* $|C_i| = 2$*, it is a binary* k*-UCS, denoted* k^b*-UCS.*

Example 3.6. *The patterns* P_1 *and* P_2 *which are learned in state-of-the-art BnB solvers, correspond respectively to a* 2^b*-UCS and a* 3^b*-UCS.*

UCS patterns have a high apparition frequency (in more than 57% of the detected ISs [Abr15]). Furthermore, certain k-UCS patterns are easily detectable by analyzing the implication graph of the obtained IS. Indeed, as outlined in the following example, the clauses which are between the conflict and the FUIP node [SS96] produce a unit resolvent clause if they are transformed by Max-SAT resolution in the reverse propagation order.

Example 3.7. *We consider the IS* $\psi = \{x_1, x_2, \overline{x_1} \vee \overline{x_2} \vee x_3, \overline{x_3} \vee x_4, \overline{x_3} \vee x_5, \overline{x_4} \vee \overline{x_5}\}$ *detected by the sequence of unit propagations represented in the form of the implication graph* G *on the left in Figure 3.4. Clearly, the node* x_3 *is the FUIP of* G*. The Max-SAT resolution transformation of* ψ *with respect to the variable sequence* $S = \langle x_5, x_4, x_3 \rangle$ *(representing the reverse order of propagation until the FUIP is encountered) is given on the right in Figure 3.4. This transformation produces the unit resolvent* $\overline{x_3}$*. Therefore, the set of clauses* $\psi' = \{\overline{x_3} \vee x_4, \overline{x_3} \vee x_5, \overline{x_4} \vee \overline{x_5}\} \subset \psi$ *is a* 3*-UCS. More specifically, since all the clauses in* ψ' *are binary, it is a* 3^b*-UCS.*

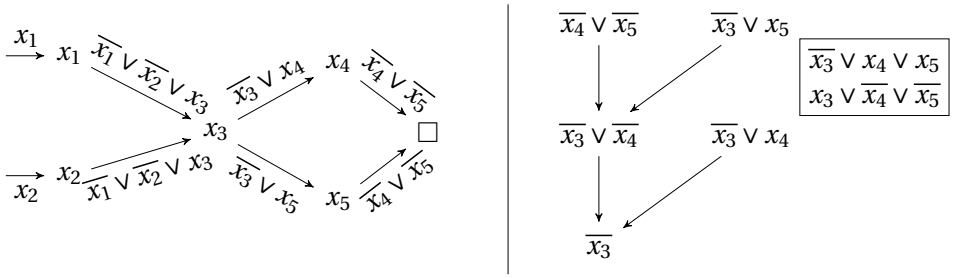


Figure 3.4: Detection (left) and transformation (right) of a UCS in Max-SAT BnB. Compensation clauses are represented in boxes.

3.2.1.5 UP-Resilience

The empirical study of UCS patterns in [AH14d] led to the first observations on the relation between Max-SAT resolution transformations and the efficiency of the SUP mechanism. These observations were formally stated by the introduction of a new property called UP-resilience [AH15b]. This new theoretical property aims to characterize Max-SAT resolution transformations in order to explain the efficiency of learning schemes observed in state-of-the-art BnB Max-SAT solvers. The main motivation behind this property is the fragmentation phenomenon [AH15b] which occurs when the information within a clause is fragmented into two (or more) clauses after transformation by Max-SAT resolution. This phenomenon, illustrated in the following example, may obstruct the exploitation of clauses by the SUP mechanism.

Example 3.8. *We consider the same IS ψ in Example 3.4 detected by the implication graph represented in Figure 3.2 and whose Max-SAT resolution transformation with respect to the the reverse order of propagation is represented in Figure 3.3 producing* $\Theta(\psi, S) = \{\square, \overline{x_1} \vee \overline{x_2} \vee x_3 \vee x_4, x_1 \vee \overline{x_3} \vee \overline{x_4}, \overline{x_1} \vee x_2 \vee \overline{x_3} \vee \overline{x_4}, x_1 \vee x_3, \overline{x_1} \vee x_2 \vee x_3, x_1 \vee x_2\}$. *If the unique neighbor of x_1 in the implication graph is set to True in the transformed IS, we obtain* $\Theta(\psi, S)|_{\{x_4\}} = \{x_1 \vee \overline{x_3}, x_1 \vee x_3, x_1 \vee x_2, \overline{x_1} \vee x_2 \vee \overline{x_3}, \overline{x_1} \vee x_2 \vee x_3\}$. *Clearly, the literal x_1 can't be propagated in* $\Theta(\psi, S)|_{\{x_4\}}$. *We can produce the resolvent x_1 if we perform a Max-SAT resolution step between the clauses $x_1 \vee x_3$ and $x_1 \vee \overline{x_3}$ but the SUP mechanism alone cannot ensure the propagation of this literal in the transformed IS even with respect to its neighborhood in the implication graph. We say that the information leading to the propagation of x_1 was fragmented into several compensation clauses.*

When fragmentation occurs, the compensation clauses which may propagate a literal of the constructed implication graph can contain additional literals which are not in its initial neighborhood. More specifically, the neighborhood of a literal l in the implication graph contain literals which are in direct interaction with it and should enable, once propagated in the detected IS, the propagation of l. However, if after the transformation of the IS, the fragmentation occurs, new literals that interact directly with l are introduced and may thus obstruct the propagation of l by SUP even when considering all the literals appearing in its neighborhood. Clearly, the power of SUP is depleted in such cases. Thus, to detect if a transformation is not affected by the fragmentation phenomenon, we can rely on the capability of SUP to propagate the literals of the constructed implication graph when their neighborhood literals are set to True. From here on, we will say that a literal l can be propagated in a formula ϕ if the unit clause $C = l$ can be inferred from ϕ by unit propagation. We provide hereafter a formal definition of UP-resilience in a given implication graph.

Definition 3.18 (UP-Resilience in an Implication Graph)**.** *Let ψ be an IS detected through the propagation steps described by an implication graph $G = (V, A)$ and S be a sequence of variables appearing in ψ. The transformation $\Theta(\psi, S)$ is UP-resilient for literal $l \in V$ in G if:*

$$\square \in neigh(l) \text{ or } l \text{ can be propagated in } \Theta(\psi, S)|_{neigh(l)}$$

$\Theta(\psi, S)$ is UP-resilient for $L \subseteq V$ in G if it is UP-resilient for each literal $l \in L$ in G and it is UP-resilient in G if it is UP-resilient for V in G.

Remark 3.2. *Note that the neighborhoods which include the special node $\square$ are not valid assignments. All transformations are considered UP-resilient for literals with such neighborhoods.*

Example 3.9. *We consider the same IS ψ in Example 3.4 and the implication graph G leading to the detection of ψ represented in Figure 3.2. In Example 3.8, we showed that the fragmentation phenomenon occurs after transformation of ψ with respect to the reverse propagation sequence S since the literal x_1 can't be propagated in* $\Theta(\psi, S)|_{neigh(x_1)}$. *Thus, the described transformation is not UP-resilient in the implication graph G.*

The previous definition of UP-resilience depends on the neighborhoods of the literals in the implication graph. However, the same IS can be detected by several sequences of

propagation steps which can be described by distinct implication graphs as illustrated in Example 3.10. To overcome this limitation, the notion of possible neighborhoods of a literal appearing in an IS is introduced in [AH15b]. We provide a formal definition below then we define the generic notion of UP-resilience which does not depend on the propagation steps leading to the IS discovery by taking all the possible implication graphs into account.

Example 3.10. *We consider the same IS* $\psi = \{x_1, x_2, \overline{x_1} \vee \overline{x_2} \vee x_3, \overline{x_3} \vee x_4, \overline{x_3} \vee x_5, \overline{x_4} \vee \overline{x_5}\}$ *in Example 3.4. In addition to the original implication graph, represented in Figure 3.2 (and also on the left in Figure 3.5), another sequence of propagations corresponding to the implication graph represented on the right in Figure 3.5 can lead to the detection of the IS* ψ*. The propagation sequences corresponding to the same implication graph are considered as equivalent.*

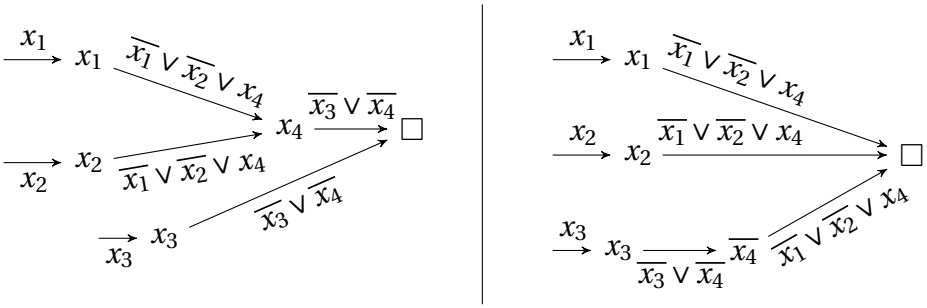


Figure 3.5: Implication graphs corresponding to the possible propagation sequences of an IS

Definition 3.19 (Possible Neighborhoods)**.** *Let* ψ *an IS and* l *be a literal appearing in* ψ*. The possible neighborhoods of* l *are defined as follows*

$$pneigh(l) = \{neigh_G(l) | G = (V, A) \text{ implication graph of } \psi \text{ s.t. } l \in V\}$$

. We naturally extend this definition on any set of literals L *appearing in* ψ *as follows:*

$$pneigh(L) = \{\bigcup_{l \in L} neigh_G(l) | G = (V, A) \text{ implication graph of } \psi \text{ s.t. } L \subseteq V\}$$

.

Definition 3.20 (UP-Resilience)**.** *Let* ψ *be an IS and* S *be a sequence of variables appearing in* ψ*. The transformation* $\Theta(\psi, S)$ *is UP-resilient for a literal* l *appearing in* ψ *if we have:*

$$\forall N \in pneigh_\psi(l) : \square \in N \text{ or } l \text{ can be propagated in } \Theta(\psi, S)|_N$$

$\Theta(\psi, S)$ *is UP-resilient if it is UP-resilient for all the literals appearing in* ψ*.*

Example 3.11. *We consider the same IS* ψ *in Example 3.4, whose distinct implication graphs corresponding to the possible propagation sequences are represented in Figure 3.5. We have*

$pneigh(x_1) = \{\{x_4\}, \{\square\}\}$. *As showcased in Example 3.8, for* $N = \{x_4\}$, *the literal* x_1 *can't be propagated in* $\Theta(\psi, S)|_N$. *Therefore, the transformation is not UP-resilient for* x_1 *and, consequently, it is not UP-resilient.*

The impact of Max-SAT resolution transformations on the SUP mechanism was studied through UP-resilience in [AH15b]. In particular, UP-resilient transformations maintain the propagations which are not necessary anymore to an IS (e.g., if it is not minimal). Indeed, if a transformation is UP-resilient for a set of literals L, then the literals of L can be propagated in the transformed formula if the literals of one of its possible neighborhoods are set to True. This is highlighted in the property established in Proposition 3.1. Furthermore, the major patterns P_1, P_2 and P_3 are UP-resilient and, thus, UP-resilience contributes to explain the empirical efficiency of learning mechanisms used in the literature [LMP07; Li+10a; AH14d; Abr15; AH15a].

Proposition 3.1 ([AH15b])**.** *Let* ψ *be an IS of and* S *be a sequence of variables appearing in* ψ. *For any set of literals* L *appearing in* ψ, *if the transformation* $\Theta(\psi, S)$ *is UP-resilient for* L *then* $\forall N \in pneigh(L) : \square \in N$ *or* $\forall l \in L, l$ *can be propagated in* $\Theta(\psi, S)|_{N \setminus \{l\}}$.

Proposition 3.2 ([AH15b])**.** *Let* ψ *be an IS. For any* $\psi' \subseteq \psi$ *that matches one of the pattern* P_1, P_2 *or* P_3. *There exists a sequence* S *of variables in* ψ' *s.t.* $\Theta(\psi, S)$ *is UP-resilient.*

Finally, it was empirically shown in [AH15b] that the order of application of Max-SAT resolution impacts the UP-resilience of the transformations by comparing two orders. The Reverse Propagation Order (RPO) applies Max-SAT resolution steps in the reverse order of propagation which is used in most of state-of-the-art solvers while the second order, called Smallest Intermediary Resolvent (SIR) and initially introduced in [AH14a], applies the Max-SAT resolution steps based on the size of the obtained resolvents, favoring the smallest ones. Furthermore, learning schemes in the literature were empirically evaluated through UP-resilience in [AH15b]. More specifically, the authors show that the learning scheme implemented in MiniMaxSat [HLO08] behaves very differently than the family of schemes relying on patterns as it learns more transformations in average (more than 60% compared to patterns) however it achieves the lowest percentage in terms of UP-resilience. In contrast, the schemes relying on patterns learn less than 20% of the transformations but achieve a high percentage of UP-resilience (100% if using a scheme relying on all the traditional patterns P_1, P_2 and P_3 and 97% if they are extended by specific UCSs as described in [AH14d]). Since the MiniMaxSat scheme achieves the worst results in terms of solved instances and average solving time, these results indicate that the UP-resilience property is a valid characterization and quality measure of Max-SAT resolution transformations in the context of Max-SAT BnB.

3.2.2 SAT-Based Approaches

SAT-based algorithms for Max-SAT rely on the power of SAT solvers as effective engines for solving decision problems [ABL13; BJM21]. These algorithms became popular this last decade as they are able to harness the efficiency of modern SAT solvers on industrial instances. In the following sections, we present the three major SAT-based algorithm families. We start by plain iterative search methods which call SAT solvers to determine the optimum value within the interval of feasible solutions. Then, we focus on core-guided algorithms which take advantage of the ability of SAT solvers to return cores when the formula is unsatisfiable. Finally, we review the Implicit Hitting Set (IHS) approach which mainly relies on a reformulation to the implicit hitting set paradigm.

3.2.2.1 Iterative Algorithms

A well-known family of SAT-based approaches to solve the Max-SAT problem consists in iterative algorithms which straightforwardly consider the decision version of Max-SAT defined below [ABL13; Mor+13; BJM21]. This decision transformation of Max-SAT simply checks whether the optimum of the input formula is bounded by a given value. Solving the Max-SAT problem for a given input CNF formula ϕ is equivalent to determining the threshold $cost \in \{0,\dots,|\phi|\}$ such that the pairs $(\phi, cost-1)$ and $(\phi, cost)$ form respectively a negative and a positive instance of the decision version. Indeed, in such case we clearly have $opt(\phi) = cost$ [20]. Iterative Algorithms for Max-SAT are based on this simple idea and mainly differ in the search methods used to explore the interval of feasible solutions $I = \{0,\dots,|\phi|\}$ in order to determine the threshold.

Definition 3.21. *The decision version of Max-SAT is defined as follows:*

> ***Decision Version of Max-SAT***
> *Input: a CNF formula* ϕ, $cost \geq 0$
> *Question:* $opt(\phi) \leq cost$?

A given bound on the optimum of a formula can be easily established by enforcing cardinality constraints [21] on a set of relaxation (or blocking) variables which are added to each clause in the formula as described in Algorithm 3.3. Each variable in R is a newly created variable (line 2) which is added to a clause in the formula (line 4). Formally, given a set of relaxation variables R and a whole number k, a cardinality constraint ensures a certain (in)equality on the sum of the variables in R and k with respect to a specific comparison operator as defined below. In particular, when $\otimes = \leq$ (resp. $\otimes = \geq$) the cardinality constraint is a typical AtMost (AtLeast) constraint on the set R. Such constraints are extensively studied in the literature and many CNF encodings have been proposed with varied degrees of efficiency [BBR06; HN13; BTS19; Ngu+20; RM21].

[20] Note that, in the case of a satisfiable formula, a simple verification with $cost = 0$ is sufficient to establish $opt(\phi) = 0$, although it is common to abuse Definition 3.21 by allowing the negative value $cost = -1$ which forms a negative instance of the decision version when coupled with any CNF formula.

[21] more generally referred to as pseudo-boolean constraints

Algorithm 3.3: Augmenting a set of clauses with relaxation variables

Input: a CNF formula ϕ
Output: (R, ϕ_R) where R is the set of relaxation variables and ϕ_R is the relaxed formula

1: **for** $C \in \phi$ **do**
2: $\quad r \longleftarrow new\_variable()$
3: $\quad R \longleftarrow R \cup \{r\}$
4: $\quad \phi \longleftarrow \phi \setminus \{C\} \cup \{C \vee r\}$
5: **return** (R, ϕ)

Definition 3.22 (Cardinality Constraint)**.** *Le R be a set of variables and k be a whole number. A cardinality constraint on R has the following form:*

$$\sum_{r \in R} r \otimes k$$

where $\otimes \in \{=, <, \leq, >, \geq\}$ is a comparison operator.

Notation 3.2. *Given a cardinality constraint ζ, we denote $CNF(\zeta)$ an encoding of ζ in CNF form.*

Clearly, augmenting a formula with relaxation variables R and a CNF encoding of an AtMost constraint enforcing a given bound $cost$ on R followed by a call to SAT solver is equivalent to solving the decision version of Max-SAT. In the following algorithms, we will denote $SAT(\phi)$ the call of a SAT solver on the input formula ϕ. The oracle returns True if the formula is satisfiable and False otherwise. Iterative algorithms rely on such a mechanism to determine the optimum of a given formula. Hereafter, we present the major iterative methods used to explore the interval of feasible solutions.

Linear Search [FM06b; ES06a; BP10] An iterative algorithm relying on linear search mainly explores the solution interval through a simple variation of k. A simple method, referred to as linear SAT-UNSAT search and described in Algorithm 3.4, consists in starting from the upper cost value $|\phi|$ and then iteratively decrementing its value by 1 until obtaining a positive instance of the decision version. The origins of this method can be traced back to the works in [Alo+02; XRS03]. In contrast, the linear UNSAT-SAT search where k is incremented until the optimum is found is mostly used, modulo certain refinements as first described in [FM06b], in the context of core-guided Max-SAT solvers which will be presented in the following section. Notice that the number of decision calls needed to solve the input formula using basic linear search is in $O(|\phi|)$.

Binary Search [FM06b; Kos+12] This method is similar to the traditional binary search algorithm performed on sorted arrays [Cor+09]. As described in Algorithm 3.5, it consists in reducing the search space interval by refining both its upper and lower bounds. In each step, the middle value of the interval is computed and provided alongside the formula as input to the decision version. If it is a positive instance the middle value becomes the new

upper bound of the interval otherwise it become the new lower bound. This treatment is repeated until the threshold value is identified. Binary search helps to reduce the number of decision calls which become bounded by $O(log(|\phi|))$. This approach was also incorporated in different core-guided algorithms [MP08; HMM11; MHM12]. In [Kos+12], the authors propose a hybrid method in which linear and binary search are alternated.

Bit-Based Search [GM07; CLS08; GM11] This approach takes advantage of the fact that the solution is a natural number. As described in Algorithm 3.6, it consists in determining the bits in the binary representation of the optimum starting from the most significant one. Similarly to binary search, this method requires $O(log(|\phi|))$ calls to a SAT solver.

Example 3.12. *We consider the formula* $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$ *where* $C_1 = x_1$, $C_2 = x_2$, $C_3 = \overline{x_2}$, $C_4 = \overline{x_1} \vee x_3$, $C_5 = \overline{x_1} \vee x_2$ *and* $C_6 = \overline{x_2} \vee \overline{x_3}$. *An iterative algorithm for Max-SAT augments the formula* ϕ *with relaxation variables to generate* $\phi_{relaxed} = \bigvee_{1 \leq i \leq 6}(C_i \vee r_i)$ *where* $R = \{r_i \mid 1 \leq i \leq 6\}$ *is a set of new relaxation variables. The execution of algorithms 3.4, 3.5 and 3.6 are summarized in Table 3.1. The returned optimum value is* $opt(\phi) = 2$.

LSU		Binary			Bit-Based		
cost	**SAT**	**cost**	**SAT**	**(cmin,cmax)**	**cost**	**SAT**	**(k,cost)**
6	True			(-1,7)			(2,0)
...	True	3	True	(-1,3)	4	True	(1,2)
2	True	1	False	(1,3)	2	False	(0,3)
1	False	2	True	(2,3)	3	True	(-1,2)

Table 3.1: Iterative Search methods for Max-SAT. The column **SAT** refers to the value returned by the oracle call on the formula $\phi_{relaxed} \cup CNF(\sum_{r \in R} r \leq cost)$ (with a strict bound in Bit search), i.e., True if the formula is satisfiable and False otherwise. For the Binary and Bit-based algorithms we additionally specify the values of the receptive pairs $(cmin, cmax)$ and $(k, cost)$ at the beginning of each iteration in the main loop.

It is easy to extend the previous methods to weighted (partial) Max-SAT. Indeed, one can easily check whether the set of hard clauses is satisfiable through a single call to a SAT oracle, in which case a search can be performed on the interval $I = \{0, \ldots, \sum_{(C,w) \in \phi \text{ s.t. } w < \infty} w\}$ through relaxation variables added to soft clauses. The SAT calls are thus performed using cardinality constraints of the following form:

$$\sum_{(C,w) \in \phi \text{ s.t } w < \infty} w_C * r_C \leq cost$$

where each soft clause C of weight w_C is augmented with the relaxation variable r_C. Major Max-SAT solvers in the literature which perform pure iterative search include MiniSat+ [ES06a], Sat4j [BP10], QMaxSAT [Kos+12] and Pacose [PRB18].

Algorithm 3.4: Linear SAT-UNSAT (LSU) algorithm for Max-SAT

Input: a CNF formula ϕ
Output: $opt(\phi)$

1: $(R, \phi) \longleftarrow Add\_Relaxation\_Variables(\phi)$
2: $cost = |\phi|$
3: **while** $SAT(\phi \cup CNF(\sum_{r \in R} r \leq cost))$ **do**
4: $\quad cost \longleftarrow cost - 1$
5: **end while**
6: **return** $cost + 1$

Algorithm 3.5: Binary search algorithm for Max-SAT

Input: a CNF formula ϕ
Output: $opt(\phi)$

1: $(R, \phi) \longleftarrow Add\_Relaxation\_Variables(\phi)$
2: $cmin = -1$
3: $cmax = |\phi| + 1$
4: **while** $cmin + 1 < cmax$ **do**
5: $\quad cost \longleftarrow \lfloor \frac{cmin+cmax}{2} \rfloor$
6: $\quad$ **if** $SAT(\phi \cup CNF(\sum_{r \in R} r \leq cost))$ **then**
7: $\quad\quad cmax \longleftarrow cost$
8: $\quad$ **else**
9: $\quad\quad cmin \longleftarrow cost$
10: **end while**
11: **return** $cmax$

Algorithm 3.6: Bit-based search algorithm for Max-SAT

Input: a CNF formula ϕ
Output: $opt(\phi)$

1: $(R, \phi) \longleftarrow Add\_Relaxation\_Variables(\phi)$
2: $k \longleftarrow \lfloor log_2 |\phi| \rfloor$
3: $cost \longleftarrow 0$
4: **while** $k \geq 0$ **do**
5: $\quad cost \longleftarrow cost + 2^k$
6: $\quad$ **if** $SAT(\phi \cup CNF(\sum_{r \in R} r < cost))$ **then**
7: $\quad\quad cost \longleftarrow cost - 2^k$
8: $\quad k \longleftarrow k - 1$
9: **end while**
10: **return** $cost$

Note that other techniques can be used to further enhance the power of iterative search. For instance, models provided by the decision oracle for satisfiable instances can be exploited to refine the upper bounds within the interval of feasible solutions as described in [BP10]. Furthermore, more relevant variations in the cost values can be performed in weighted Max-SAT by ensuring the validity of certain bounds as suggested in [ABL10a]. Finally, large formulas require the addition of many relaxation variables and the generation of CNF encodings of cardinality constraints with large size which may render pure iterative search inefficient in such cases. In the next section, we present a second family of SAT-based approaches, called core-guided algorithms, which try to offset this disadvantage by further exploiting the power of modern SAT solvers.

3.2.2.2 Core-Guided Algorithms

The core-guided approach for Max-SAT solving regroups a family of algorithms which take advantage of the ability of SAT solver to return cores when the input formula is unsatisfiable [ABL13; Mor+13; BJM21]. Since these algorithms often maintain a set of hard clauses, we present them in the context of partial Max-SAT while supposing that the set of hard clauses is initially satisfiable. The first core-guided algorithm for Max-SAT was introduced in [FM06b; Fu07]. This procedure, commonly referred to as the Fu and Malik (FM) Algorithm, is described in Algorithm 3.7. It consists in adding cardinality constraints solely on the clauses of the core returned by the SAT solver. More specifically, a call to a SAT oracle on the current formula is performed in each step (line 3). If the formula is unsatisfiable, the solver returns a core in the form of a subset of clauses in the formula. Relaxation variables are then added to these clauses and an at most constraint enforcing a bound of one is encoded and added to the set of hard clauses (lines 6-8). Furthermore, the optimal cost is incremented by 1 (line 9) until the formula becomes satisfiable, in which case it is returned by the algorithm as the optimum of the input formula (lines 4-5).

Algorithm 3.7: Fu and Malik Algorithm

Input: a partial CNF formula $\phi = H \cup S$
Output: $opt(\phi)$

1: $cost \longleftarrow 0$
2: **while** $True$ **do**
3: $\quad (solved, \psi) \longleftarrow SAT(H \cup S)$
4: $\quad$ **if** $solved$ **then**
5: $\quad\quad$ **return** $cost$
6: $\quad (R, \psi') \longleftarrow Add\_Relaxation\_Variables(\psi \cap S)$
7: $\quad S \longleftarrow S \setminus (S \cap \psi) \cup \psi'$
8: $\quad H \longleftarrow H \cup CNF(\sum_{r \in R} r \leq 1)$
9: $\quad cost \longleftarrow cost + 1$
10: **end while**

Example 3.13. *We consider the partial CNF formula* $\phi = H \cup S$ *where* $H = \emptyset$ *and* $S = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$ *where* $C_1 = x_1$, $C_2 = x_2$, $C_3 = \overline{x_2}$, $C_4 = \overline{x_1} \vee x_3$, $C_5 = \overline{x_1} \vee x_2$ *and* $C_6 = \overline{x_2} \vee \overline{x_3}$. *The Fu and Malik algorithm calls a SAT oracle on* ϕ *which returns that the formula is unsatisfiable alongside, for instance, the core* $\psi = \{C_2, C_3\}$. *Note however that the returned core is not necessarily minimal in terms of size. The algorithm adds relaxation variables to core* ψ *to obtain* $\psi' = \{x_2 \vee r_1, \overline{x_2} \vee r_2\}$ *where* r_1 *and* r_2 *are new relaxation variables. Since* ψ *contains soft clauses only, the set* S *is updated as follows* $S = S \setminus \psi \cup \psi'$. *The cardinality constraint is also added to the set of hard clauses, i.e.,* $H = CNF(r_1 + r_2 \leq 1)$, *and the cost value is incremented by 1. A second SAT call is then performed. The formula is still unsatisfiable and the oracle returns the core* $\psi = \{C_1, C_4, C_5, C_6\}$. *Similarly to the above treatment, we add relaxation variables to* ψ *and the relaxed set* $\psi' = \{C_1 \vee r_3, C_4 \vee r_4, C_5 \vee r_5, C_6 \vee r_6\}$ *is used to update the set of soft clause* S *while adding the encoding* $CNF(r_3 + r_4 + r_5 + r_6 \leq 1)$ *to the set of hard clauses. After incrementing the cost, a last call to the SAT oracle returns that the formula is satifiable and the optimum value* 2 *is returned by the FM algorithm.*

Many variations of this algorithm have been proposed in the literature. In particular, a weighted version of FM, the WPM1 algorithm, was introduced in [ABL09b] [22]. The idea is similar to FM except that the clauses are replicated with respect to the minimum weight in the core. This treatment corresponds to application of the unfold rule in Definition 3.13. Formally, let $minw$ denote the minimum weight over the clauses in the core ψ, i.e., $minw = min_{C \in \psi} w(C)$. For each soft clause C in ψ of weight higher than $minw$, a copy is generated with weight $w(C) - minw$. Then, relaxation variables are added to all soft clauses in the core, which now have weight $minw$. A cardinality constraint is added to the set of hard clauses as in FM. Finally, the value $minw$ is used to update the cost (instead of incrementing it by 1).

Other variations of FM include (W)MSU3 [MP07], (W)PM2 [ABL09a; ABL09b; ABL10b] and OLL [MDM14] among many others [MM08; MP08; HMM11; ADG15; ADR15; IMM19; IBJ21]. The MSU3 algorithm adds relaxation variables on demand as in FM but maintains one variable only for each clause [MP07]. More specifically, when a core is found, the soft clauses which were not relaxed in prior steps are augmented with new relaxation variables and the bound enforced on the relaxation variables is updated. This algorithm, implemented in the MSCG [MIM14] and Open-WBO [23] [MML14] solvers, is very similar to iterative linear search but relaxation variables are added depending on the cores as is expected of a core-guided approach. The PM2 algorithm similarly maintains one relaxation variable per clause but allows to split disjoint cores over different cardinality constraints [ABL09b; ABL09a; ABL10b]. To this end, PM2 maintains a record of each core found so far along with their sets of relaxation variables. When a new core disjoint with the previous ones is found, new relaxation variables are added to its soft clauses and an additional AtMost constraint is enforced over its variables. Otherwise, the previous constraints which clash with it are merged into a new AtMost constraint enforcing the sum of bounds (plus one for the current core) over their relaxation variables and the additional variables necessary to relax soft clauses in the core.

[22] A similar algorithm, called WMSU1, was also introduced independently in [MSP09].
[23] http://sat.inesc-id.pt/open-wbo/

A more sophisticated core-guided algorithm introduced in the literature for Max-SAT solving is OLL [MDM14] [24] which is implemented in the MSCG [MIM14] and RC2 [IMM19] solvers. The main idea behind OLL is to reuse cardinality constraints as they are discovered while mixing the strengths of previously introduced methods. Similarly to MSU3, OLL adds one relaxation variable per clause while also adding a new cardinality constraint for each core as in FM. The main difference between OLL and these algorithms is that the soft clauses are hardened after relaxation while cardinality constraints are added as unit soft clauses. More specifically, whenever a new core is found, the working formula is updated with a new soft constraint allowing to set one newly introduced relaxation variable to True if all the previous soft constraints in the core are satisfied. Furthermore, the bounds enforced on the previous constraints are also increased by 1. Note that this algorithm takes advantage of well-known encodings which allow to independently encode the sum side in a cardinality constraint and to enforce a specific bound through a single literal [BB03; Sin05; ES06b; Mar+14a].

Unlike the previously described algorithms which use cardinality constraints through the addition of relaxation variables, other core-guided algorithms in the literature use pure Max-SAT inference to transform cores returned by SAT solvers. Similarly to BnB solvers, these methods rely on the Max-SAT resolution rule in Definition 3.11 to transform the cores returned by the SAT oracle. The first attempt in [HM11] using such a method consists in retrieving a resolution proof whenever a core is returned by the oracle [25]. If the proof is read once, it is simply adapted into a Max-SAT resolution proof otherwise a traditional core-guided algorithm such as FM is applied. To adapt the read-once resolution proof, one only needs to replace each resolution step by a Max-SAT resolution step [26] [BLM06; BLM07]

Algorithm 3.8 describes another inference-based core guided procedure called PMRes which was implemented in the EVA solver [NB14]. Whenever a core ψ is returned by the SAT solver, PMRes renames the soft clauses by introducing equivalent literals (lines 5-9). Let R be the set containing the negation of such literals. Since at least one soft clause must be falsified, the clause $C = \bigvee_{r \in R} r$ can be added as a hard clause to ensure this context-induced constraint while maintaining Max-SAT reducibility [27]. Then, a sequence of Max-SAT resolution applications is performed in **Apply_MaxSAT_Resolution**(H, S, R) (line 6) and the transformed set of hard and soft clauses are returned after this procedure. More specifically, at each step, a literal l is chosen in R and a Max-SAT resolution step between clauses $\overline{l}$ and $\bigvee_{r \in R} r$ is performed while adding the generated resolvent clause and compensation clauses to the set of soft clauses. The literal l is then deleted from l and the next step is performed with respect to the updated set R. Note that the premises of the first step corresponds to $\overline{l}$ and C [28] while the conclusion of the last step is the empty clause, in which case the cost is updated to account for it instead of adding it to the formula (lines 13).

[24] OLL was originally introduced in the context of Answer Set Programming (ASP) [And+12].
[25] refer to Section 2.3
[26] refer to Section 3.3.2 for more details.
[27] refer to Definition 3.9
[28] C can be considered as a soft clause in this step since hard clauses can be used as a premise in the Max-SAT resolution rule multiple times without being deleted. In the weighted case, this corresponds to the scission of an infinite weight clause to generate a clause of weight 1 using the unfold rule in Definition 3.13

Algorithm 3.8: PMRes Algorithm

Input: a partial CNF formula $\phi = H \cup S$
Output: $opt(\phi)$

```
1:  cost ⟵ 0
2:  while True do
3:      (solved, ψ) ⟵ SAT(H ∪ S)
4:      if solved then return cost
5:      for C ∈ ψ ∩ S do
6:          r ⟵ new_variable()
7:          R ⟵ R ∪ {r}
8:          H ⟵ H ∪ CNF(C ⇔ r̄)
9:          S ⟵ S \ {C} ∪ {r̄}
10:     end for
11:     H ⟵ H ∪ {⋁_{r∈R} r}
12:     H, S ⟵ Apply_MaxSAT_Resolution(H, S, R)
13:     cost ⟵ cost + 1
14: end while
```

It is easy to notice that, if a new literal d such that $d \Leftrightarrow \bigvee_{r \in R \setminus \{l\}} r$ is introduced in each step, then the compensation clauses generated by Max-SAT resolution between $\overline{l}$ and $\bigvee_{r \in R} r$ can be compressed into the form $\overline{l} \vee \overline{d}$. The PMRes algorithm employs this mechanism by introducing new literals in the Max-SAT resolution transformation process and adding the required equivalences to the set of hard clauses.

Example 3.14. *We consider the partial CNF formula $\phi = H \cup S$ where $H = \emptyset$ and $S = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$ where $C_1 = x_1$, $C_2 = x_2$, $C_3 = \overline{x_2}$, $C_4 = \overline{x_1} \vee x_3$, $C_5 = \overline{x_1} \vee x_2$ and $C_6 = \overline{x_2} \vee \overline{x_3}$. The PMRes algorithm calls a SAT oracle on ϕ which returns that the formula is unsatisfiable alongside the core $\psi = \{C_2, C_3\}$. Two new variables r_1 and r_2 are introduced to represent the soft clauses in the core and we update the hard and soft clause sets as follows: $H = CNF(C_2 \Leftrightarrow \overline{r_1}) \cup CNF(C_2 \Leftrightarrow \overline{r_1}) \cup \{r_1 \vee r_2\}$ and $S = S \setminus \psi \cup \{\overline{r_1}, \overline{r_2}\}$. Under a lexicographic ordering of the variables, the Max-SAT resolution rule is applied first on the clauses $r_1 \vee r_2$ and $\overline{r_1}$ to generate the resolvent clause r_2 and the compensation clause $\overline{r_1} \vee \overline{r_2}$ then on clauses r_2 and $\overline{r_2}$ to generate the empty clause and thus the cost is incremented by 1. Consequently, after the transformation we have $S = \{C_1, C_4, C_5, C_6, \overline{r_1} \vee \overline{r_2}\}$ and $cost = 1$. The SAT oracle is called again on $H \cup S$ and returns the core $\psi = \{C_1, C_4, C_5, C_6\}$. We update the sets of hard and soft clauses as follows $H = H \cup CNF(C_1 \Leftrightarrow \overline{r_3}) \cup \cdots \cup CNF(C_6 \Leftrightarrow \overline{r_6}) \cup \{r_3 \vee r_4 \vee r_5 \vee r_6\}$ and $S = S \setminus \psi \cup \{\overline{r_3}, \overline{r_4}, \overline{r_5}, \overline{r_6}\}$. As in the previous transformation, several Max-SAT resolution steps are performed to obtain the empty clause, which we represent in Figure 3.6 (in lexicographic order). The transformation produces the set $S = \{\overline{r_1} \vee \overline{r_2}, \overline{r_3} \vee \overline{r_4}, \overline{r_3} \vee r_4 \vee \overline{r_5}, \overline{r_3} \vee r_4 \vee r_5 \vee \overline{r_6}, \overline{r_4} \vee \overline{r_5}, \overline{r_4} \vee r_5 \vee \overline{r_6}, \overline{r_5} \vee \overline{r_6}\}$ while the cost is also incremented by 1 to account for the empty clause. A last call to the sat oracle returns that the formula is satisfiable and thus the algorithm terminates specifying that the optimum value is 2. Note that the compensation clauses can also be compressed as specified in [NB14] by introducing additional variables. For instance, if we introduce a new variable d with the hard encoding $CNF(d \Leftrightarrow x_4 \vee x_5 \vee x_6)$*

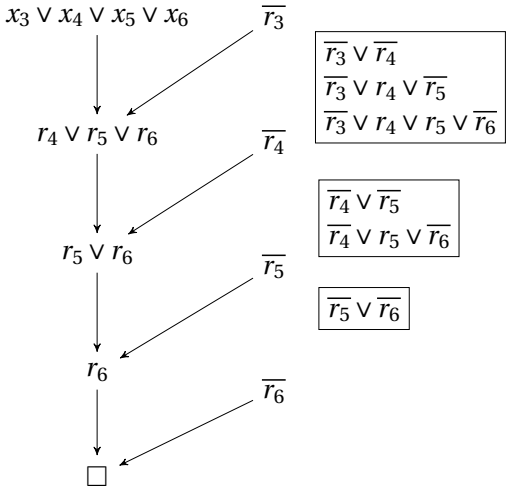


Figure 3.6: Max-SAT resolution transformation of core in PMRes. Compensation clauses for each step are represented in boxes.

the compensation clauses generated in the first step in figure 3.6 can be compressed in the form of a single clause $\overline{x_3} \vee \overline{d}$.

Finally, we briefly recall some specific metaheuristics and techniques used in core-guided solvers [29]. One major technique used in the context of weighted Max-SAT is stratification [Ans+13]. This technique aiming to generate cores with higher minimum weights consists in partitioning the soft clauses into k sets such that the clauses in set $i \in \{2, \ldots, k\}$ have high weight than those in the set $i-1$. The solving process starts by considering only the set k with the highest weights, ignoring all the other soft clauses, thus potentially ensuring the detection of better cores in terms of minimal weight. Once the problem restricted to the higher-weight clauses is solved, the search process is restarted by adding the next set until all the soft clauses become included in the working formula. The boolean lexicographic optimization approach in [Mar+11] can also be helpful in this regard.

An important feature of SAT-based algorithms for Max-SAT is that they rely on the power of assumption-based SAT solving. Modern SAT solvers often provide interfaces for solving formulas where assumptions are provided in the form of a set of literals. This feature originates from the work in [ES03b] and is now commonly used in modern SAT solvers. When the solver is called on a given formula, it is forced to assign the assumption literals before starting the actual search. If a new literal l is added to a clause C before passing it to the SAT oracle, then adding l as an assumption can render the clause inactive (since it will be satisfied by l) while assuming $\overline{l}$ can activate the clause for the subsequent search to be performed by the SAT solver. Assumptions are a powerful tool that can help Max-SAT solvers take advantage of incremental SAT solving where a single instance of the oracle is used to solve all of the formulas while maintaining information acquired throughout the the previous calls. In the context of Max-SAT, such a mechanism was first employed and

[29] Note that some of the mentioned techniques can also be applied in other SAT-based approaches.

shown effective in [Mar+14b]. A restart mechanism was proposed in [Si+16] enabling to discard learned information if deemed useful. More recently, Hickey and Bacchus showed that two light-weight techniques, namely enqueuing assumptions in bulk and enhancing trail-savings, can entail non-trivial performance improvement in the context of incremental SAT-based Max-SAT solving [HB19].

Another mechanism used in modern core-guided solvers is core exhaustion [Ans+13; IMM19]. The idea behind this heuristic is to quickly increase the bound within an At-Most constraint on the relaxation variables of a core. More specifically, given a newly identified and relaxed core ψ alongside its set of relaxation variables R, the formula $H \wedge \psi \wedge CNF(\sum_{r \in R} r \leq 1)$ (where H denotes the set of hard clauses) may still be unsatisfiable. Successive incremental calls to the SAT solver focusing on the last core may therefore enable a cost-effective improvement of the bound on the cardinality constraint. A different mechanism which is also specific to cores, and which simply aims to reduce their size, is core reduction. The first reduction technique, called core trimming, was used in [MIM14] by performing successive SAT calls on each detected core alongside the set of hard clauses until a fixed point (in terms of core size) is reached. Another reduction mechanism used in [IMM19] is based on the deletion-based minimal core extraction method introduced in [Sil10] while limiting the total number of conflicts encountered during the successive SAT calls.

3.2.2.3 Implicit Hitting Set Approach

The implicit Hitting Set (IHS) approach for Max-SAT described in Algorithm 3.9 was initially introduced in [DB11; Dav13a]. IHS-based Max-SAT solving also relies on the ability of SAT solvers to return cores. However, cores are stored in a list K and are exploited differently then the core-guided approaches presented in the previous section. Indeed, as its name suggests, this approach is based on the implicit hitting set paradigm [DCB10a; Cha+11]. Formally, given a set of cores K, a hitting set HS of K is a set of soft clauses such that for all $\psi \in K$ we have $HS \cap \psi \neq \emptyset$. A minimal hitting set HS of K is a hitting set of K with minimal size, i.e., $|HS| \leq |HS'|$ for any hitting setting HS' of K. Clearly, the size of a minimal hitting set HS of K such that a call to a SAT solver on $\phi \setminus HS$, where ϕ is the input formula, corresponds to the optimum of ϕ since the clauses in the minimal set HS are sufficient to cover all the cores in K. Note how this approach does not include any treatment on the detected cores nor on the input formula which is why it is usually considered as a separate approach, different from the core-guided algorithms described in the previous section.

Example 3.15. *We consider the CNF formula* $\phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$ *where* $C_1 = x_1$, $C_2 = x_2$, $C_3 = \overline{x_2}$, $C_4 = \overline{x_1} \vee x_3$, $C_5 = \overline{x_1} \vee x_2$ *and* $C_6 = \overline{x_2} \vee \overline{x_3}$. *The set of detected cores is initially empty, i.e.,* $K = \emptyset$, *and thus a minimal hitting set on* K *is clearly* $HS = \emptyset$. *The IHS-based algorithm thus calls a SAT oracle on* ϕ *which returns that the formula is unsatisfiable alongside the core* $\psi_1 = \{C_2, C_3\}$. *The set of detected cores is thus updated as follows:* $K = \{\psi_1\}$. *The algorithm then computes a minimal hitting set on* K. *For instance* $HS = \{x_2\}$ *is such a set as the clause* x_2 *is sufficient to cover the only core in* K. *Next, the algorithm calls the SAT solver on the formula* $\phi \setminus \{x_2\}$. *The core* $\psi_2 = \{C_1 \wedge C_4 \wedge C_5 \wedge C_6\}$ *is returned and added to the set* K. *Now, a minimal hitting set on* K *is for example* $HS = \{x_1, x_2\}$ *as clauses* x_1 *and* x_2 *are*

Algorithm 3.9: Implicit Hitting Sets for Max-SAT

Input: a CNF formula ϕ
Output: $opt(\phi)$

```
1: K ⟵ ∅
2: while True do
3:     HS ⟵ Min_Hitting_Set(K)
4:     (solved, ψ) ⟵ SAT(φ \ HS)
5:     if solved then
6:         return |HS|
7:     K ⟵ K ∪ ψ
8: end while
```

clearly sufficient to cover ψ_1 and ψ_2 and no better hitting set, with respect to size, exists. A last call to the SAT oracle is performed on $\phi \setminus HS$, which returns that formula is satisfiable and therefore the algorithm returns the optimum value $|HS| = 2$.

Different methods can be used to compute minimal-size hitting sets including BnB [DB11; Blä+22] and SAT-based methods [Ign+15]. However, it is more common to reduce this problem to an Integer Linear Programming (ILP) instance which is solved by a dedicated solver [DB11; Dav13a; SBJ16]. Extensions to the basic IHS approach described in Algorithm 3.9 include maintaining lower and upper bounds throughout the search as is the case in binary iterative search [DB13b]. Other mechanisms aiming to enhance the efficiency of the ILP component were introduced in the literature such as additional constraints extracted from the formula [DB11; DB13a] or employing bounding techniques to harden clauses [Bac+17]. Recent work also addresses the large worst-case number of cores that IHS solvers may have to extract before terminating by introducing the notion of abstract cores, which enables a compact representation for a potentially exponential number of regular cores [BBP20b]. Solvers based on the IHS approach include MaxHS [30] [DB11; Dav13a], which invokes the CPLEX solver [31] for the ILP optimization component, and LMHS [32] [SBJ16], which includes interfaces to two ILP solvers, namely CPLEX and SCIP [33] [Ach09].

3.2.3 Other Complete Approaches

In this section, we briefly overview other complete methods that were not accounted for in previous sections. One such approach which was investigated in the literature is reducing Max-SAT to other well-known optimization problems. Note that it is common to use reformulations and relaxations for specific components in Max-SAT algorithms as is the case for the IHS SAT-based approach presented in the previous section. Full encodings

[30] http://www.maxhs.org/
[31] https://www.ibm.com/products/ilog-cplex-optimization-studio
[32] https://www.cs.helsinki.fi/group/coreo/lmhs/
[33] https://www.scipopt.org/

from Max-SAT to other problems have also been introduced in the literature although with less success compared to the current methods dedicated for Max-SAT. Known reductions include modeling Max-SAT as an ILP program which is passed to a dedicated solver such as CPLEX or SCIP [AG13]. Other paradigms used in the literature include Answer Set Programming (ASP) [OJ09] and Constraint Programming (CP) [De +03].

Complete methods for Max-SAT also include sequential portfolio algorithms which take advantage of the diversity in the Max-SAT solving landscape by predicting the suitability of different solvers for specific instances. To this end, features such as problem size, balance and local search probes [34] were used to approximate the runtime of a solver on a particular instance through linear regression in [Mat+08]. A classifier based on a cost-sensitive hierarchical clustering model, able to handle larger feature sets, was also devised in [Mal+13]. The most recent algorithm in this category is the instance-specific algorithm configurator introduced in [AMS14; Ans+16] and based on the ISAC clustering method in [Kad+10]. Finally, as in SAT [35], different parallel algorithms were also devised for Max-SAT in the literature including (parallel) portfolios and space-splitting methods which perform an exhaustive exploration of the search space [LMM18].

3.2.4 Beyond Complete Methods

In addition to complete approaches for Max-SAT solving, state-of-the-art methods also include incomplete algorithms. Incomplete algorithms for Max-SAT mainly rely on Stochastic Local Search (SLS) as in SAT [36]. In fact, SLS algorithms for SAT, modulo some minor variations, can be used for Max-SAT. Indeed, the aim of Max-SAT SLS methods is to find a good cost value that can be achieved by an assignment of the input formula and which is as close as possible to its optimum value. As such, the value of the objective function in Definition 2.11 is saved and updated whenever it can be improved during a traditional SLS search as described Algorithm 2.3. This value is then returned by the algorithm when all the tries are done. Historically, the first SLS algorithm devised specifically for Max-SAT can be traced back to the work in [HJ90]. The traditional SLS architechtures, i.e., GSAT and Walk-SAT, can be applied in the context of Max-SAT as shown in [SKC94]. More complex methods were adapted or tailor-made for Max-SAT. In [YI98; YI01], the notion of neighborhood is extended to encompass all the possible assignments when simultaneously flipping two or three variables. History-based metaheuristics, such as Tabu Search and Configuration Checking (CC), were also adapted for Max-SAT [SHS03; BD05; Luo+15; CJS15; Luo+17a]. Different dynamic search techniques [HTH02; Cai+16; LC18; CL20; LC20] and adaptive search mechanisms [LH12; AM21] were similarly tailored for Max-SAT. Note that in the context of weighted (partial) Max-SAT, the weighted cost in Definition 3.6 can be used, although it is more common in recent SLS solvers to distinguish between soft and hard clauses by considering different cost functions and/or using different mechanisms favoring the satisfaction of hard clause [Cai+14; Luo+15; Luo+17a; Cai+16; LC18; CL20; LC20]. More recently, Zheng et al. proposed a Multi-Armed Bandit (MAB) framework in [Zhe+22] to help

[34] These features are detailed in [Nud+04].
[35] refer to Section 2.2.1.4
[36] refer to Section 2.2.2.1 on SLS for SAT

escape local optima by selecting a relevant arm within the set of soft clauses. Competitive SLS solvers for Max-SAT include CCLS [Luo+15], Dist [Cai+14; Cai+16] and SATLike [LC18; CL20] among others.

Approximation algorithms for Max-SAT have also been extensively studied in the literature. Such algorithm must be efficient while providing approximate solutions to optimization problems with certain guarantees. Formally, an approximation algorithm for a maximization problem is a polynomial-time algorithm A such that for any instance x there exists an approximation factor $\rho < 1$ providing a guarantee on the returned solution $s = A(x)$ as in $\rho.s^* \leq s$ where s^* denote the optimal solution on input x. In the context of Max-SAT, we expect such algorithms to provide a solution $cost$ such that for any input formula ϕ we have $cost \leq \delta.opt(\phi)$ where $\delta = \frac{1}{\rho}$. The first approximation algorithms for Max-SAT were based on simple greedy methods and achieved a guarantee factor $\delta = \frac{1}{2}$ [Joh73; Joh74]. More refined algorithms relying on network flow techniques [Yan92; Yan94] and ILP relaxation of Max-SAT [GW94b] enabled to provide better guarantees with a factor of $\frac{3}{4}$. In [Hås97; Hås01], Håstad showed that it is not possible to achieve a factor better than $\frac{7}{8}$ unless P=NP. This bound was achieved by Karloff and Zwik in [KZ97] through a semidefinite programming reformulation of Max-3-SAT. More recent works in this context focus on proving approximation results for specific variants of Max-SAT or on devising simpler algorithms for approximating Max-SAT and its variants [AW02; ABZ05; MNH08; Zuy11; PWZ14]. Finally, an empirical evaluation of fast approximation algorithms for Max-SAT is conducted in [PW16; PW17] showing that such algorithms can deliver very good solutions at low computational cost and that a hybrid scheme combining the strengths of such algorithms with SLS could provide better cost-effective solutions. Note however that classical approximation algorithms are not used in a competitive context. A recent incomplete paradigm that was used in competitive incomplete solvers, such as Open-WBO-Inc [Jos+18; Jos+19] and LinS-BPS [DS19], relies on solution approximation through iterative search without any formal guarantees.

3.3 Proofs and Certificates for Max-SAT

In this section, we are interested in certificates and proof systems for the Max-SAT problem. We start by defining some necessary notions and overviewing proof systems for Max-SAT. Then, we focus on Max-SAT Resolution [Rob65] as a proof system for Max-SAT, which is one of the first and most studied systems in the literature. We particularly recall some known results on poof adaptations and completeness of Max-SAT resolution which will motivate our contributions in Chapter 6.

3.3.1 Proof Systems for Max-SAT

Similarly to SAT, a certificate of optimality for maximum satisfiability is a proof that the cost value returned by a Max-SAT solver is indeed the optimum of the formula and this notion is mainly studied through the lens of proof systems. However, unlike SAT, such certificates have been marginally studied and generated in practice. Therefore, we focus

hereafter on reviewing proof systems for Max-SAT whose study has seen a surge of interest these last years. Formally, a proof system for Max-SAT is a set of inference rules defined by their antecedents and conclusions, both usually expressed in the form of a set of clauses. A major difference compared to SAT inference rules is that the conclusions replace the antecedent clauses if they are within the formula. This is necessary to ensure the soundness of the rules which is ensured by Max-SAT equivalence introduced in Definition 3.8. We maintain the same notations introduced in Section 2.3.1. The definitions of soundness and completeness where, Max-SAT equivalence is considered instead of SAT equivalence, also remain valid for Max-SAT. We provide below a definition of refutational completeness which is valid for Max-SAT.

Definition 3.23 (Refutational Completeness)**.** *A proof system S for Max-SAT is refutationally complete if, for any formula ϕ, $\phi \vdash_S \underbrace{\square \wedge \cdots \wedge \square}_{opt(\phi)} \wedge \phi'$ where ϕ' is a satisfiable CNF formula.*

Many proof systems for Max-SAT have been introduced and studied in the literature. One of the first and most studied systems is Max-SAT resolution [Rob65], which we will overview in the following sections. Other known systems include tableau calculus [LMS16], dual-rail [IMM17; Bon+18], circular proofs [AL19; BL20] and subcube sums [Fil+20]. Similarly to SAT, the relative strengths of proof systems for Max-SAT is compared through the notion of simulation in Definition 2.27 and was investigated in recent works specifically with respect to Max-SAT resolution and its extensions [LR20a; LR20b; BL20]. In our work in Section 6.5.2, we also consider a the more generic notion of simulation, defined below, which we refer to as inferential simulation. It differs from the traditional definition of simulation in Definition 2.27 [37], in the sense that it does not restrict the proofs to refutations. Clearly, if P_1 i-p-simulates P_2 then P_1 also r-p-simulate P_2 but the opposite is not necessarily true.

Definition 3.24 (Inferential Polynomial Simulation)**.** *Let S and S' be two proof systems. S inferentially polynomially simulates (i-p-simulates) S' if there exists a polynomial computable function f such that for any proof π in S' deducing C from ϕ, $f(\pi)$ is a proof deducing C from ϕ in S. S and S' are i-p-equivalent if S i-p-simulates S' and S' i-p-simulates S.*

3.3.2 Max-SAT Resolution Calculus

One of the first and most studied proof systems for Max-SAT is the Max-SAT resolution calculus (MaxRes) which relies on an inference rule extending resolution for Max-SAT [LH05a; BLM06; BLM07; LHG08] [38]. Other than the resolvent clause, this rule, called Max-SAT resolution, introduces new clauses referred to as compensation clauses and essential to preserve Max-SAT equivalence. We provide below a more compact definition of this rule relying on the following rewriting:

$$C \vee \overline{a_1 \vee a_2 \vee \ldots \vee a_n} = (C \vee \overline{a_1}) \wedge (C \vee a_1 \vee \overline{a_2}) \wedge \ldots \wedge (C \vee a_1 \vee a_2 \vee \ldots \vee \vee \overline{a_n})$$

[37] which we refer to as refutational polynomial simulation (r-p-simulation) in Section 6.5.2 to avoid confusion
[38] refer to Definition 3.11

This rewriting was introduced in [LHG08] as a recursive rule to transform the compensation clauses into CNF form. This also entails that the Max-SAT resolution rule depends on the ordering of the literals, as reported in [BLM07; LHG08]. Max-SAT resolution plays an important role in the context of Max-SAT theory and solving as it is extensively used and studied in the context of BnB algorithms for Max-SAT [LMP07; Küe12; AH14b; AH15a; AH15b; Abr15; CH19] and more marginally in the context of core-guided approaches [HM11; NB14] [39].

Definition 3.25 (Max-SAT Resolution [LH05a; BLM06; BLM07; LHG08])**.** *Given two opposed clauses C_1 and C_2, the Max-SAT resolution rule is defined as follows:*

$$\frac{C_1 = x \vee A \qquad C_2 = \overline{x} \vee B}{\begin{array}{c} C_r = A \vee B \\ CC_1 = x \vee A \vee \overline{B} \\ CC_2 = \overline{x} \vee \overline{A} \vee B \end{array}}$$

where C_r is the resolvent clause and CC_1, CC_2 are compensation clauses.

A Max-SAT resolution proof or derivation of a formula ϕ' (resp. clause C) from ϕ is a finite sequence of Max-SAT resolutions starting from the clauses of ϕ and deducing ϕ' (resp. such that $C \in \phi'$) and is usually represented as a finite sequence of formulas. A Max-SAT resolution proof can be represented as a bipartite DAG whose nodes are either clauses or inference steps (in which case they will be omitted for more simplicity). We maintain the size and width measure introduced in Section 2.3.1. As a proof system, Max-SAT resolution is sound and refutationally complete for Max-SAT [BLM06; BLM07]. In particular, for a given CNF formula ϕ, it is possible to generate a Max-SAT resolution proof of its optimum by applying the saturation algorithm [BLM07] which consists in successively saturating each variable x by applying Max-SAT resolution on x until all pairs of clauses containing x are opposed on another variable. Using saturation it is possible to provide a Max-SAT resolution proofs of the optimum of a given formula ϕ containing $O(|\phi| \times n \times 2^{|var(\phi)|})$ inference steps as established in the following theorem.

Theorem 3.2 ([BLM06; BLM07])**.** *Let ϕ be a CNF formula. We can deduce $\phi \vdash_{MaxRes} \underbrace{\square \wedge \cdots \wedge \square}_{opt(\phi)} \wedge \phi'$ where ϕ' is satisfiable in $O(|\phi| \times |var(\phi)| \times 2^{|var(\phi)|})$ inference steps.*

Example 3.16. *We consider the formula $\phi = (x_1) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$. The saturation algorithm applied in lexicographic order ($x_1 < x_2 < x_3$) on ϕ enables to deduce $\phi \vdash_{MaxRes} \square \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$. The corresponding proof is represented in Figure 3.7.*

[39] refer to Section 3.2 for more details.

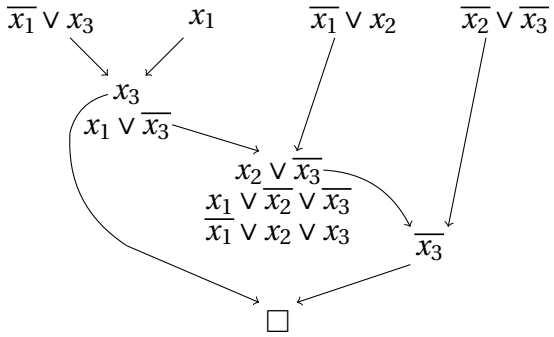


Figure 3.7: A Max-SAT resolution proof

Note that Max-SAT resolution is not (inferentially) complete as shown in Proposition 3.3. In recent work, Max-SAT resolution was augmented with the split rule, defined below, forming the system *ResS* [LR20b; BL20; Fil+20]. It was shown that the addition of this rule to Max-SAT resolution is sufficient to generate a complete stronger proof system for Max-SAT. Note that ResS is stronger than MaxRes in the sense that the former p-simulates the latter while the opposite simulation is not possible. This result is established in [LR20b] by relying on a variation of the Pigeon Hole Problem (PHP) problem [40].

Proposition 3.3 ([LR20b])**.** *Max-SAT resolution is not complete.*

Proof. Consider the formula $\phi = (x_1) \wedge (x_2)$. Clearly, we have $\phi \equiv (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2}) \wedge (x_2)$. However, $\phi \nvdash_{MaxRes} (x_1 \vee x_2)$.

■

Definition 3.26 (Split)**.** *Given a clause C and a variable x, the split rule is defined as follows:*

$$\frac{C}{C \vee x \qquad C \vee \overline{x}}$$

Theorem 3.3 ([LR20b])**.** *ResS is complete.*

Unlike resolution, the Max-SAT resolution rule replaces the premises by the conclusions. Larrosa et al. describe Max-SAT resolution as "*a movement of knowledge*" [LHG08]. Because of this specificity, it is not easy to adapt a resolution proof to obtain a Max-SAT resolution proof. Indeed, in resolution proofs, several resolution steps can share the same premise, because the premises are not consumed after the application of a resolution step. On the other hand, the premises of a Max-SAT resolution step are consumed after its application. Consequently, the immediate adaptation of a resolution proof for Max-SAT is only possible if it is read-once [BLM06; BLM07]. Yet, this simple observation was exploited to devise the SAT based algorithm described in [HM11]. Our contributions in Section 6 will aim to tackle the open question of whether this result could be extended to other resolution classes, potentially by augmenting Max-SAT resolution with other rules. In this context, a max-refutation will refer to a sound Max-SAT proof, using Max-SAT resolutions and potentially

[40] In PHP, the goal is to assign $m+1$ pigeons to m holes without any pair of pigeons sharing their hole.

splits, deducing the empty clause. It is important to distinguish between max-refutations and Max-SAT resolution refutations as the former may rely on additional rules (mainly the split rule) while the latter relies exclusively on Max-SAT resolution.

Proposition 3.4 ([BLM06; BLM07])**.** *Max-SAT resolution p-simulates read-once resolution.*

Proof. Clearly, every read-once resolution proof can be trivially adapted to Max-SAT proofs by replacing each resolution step with a Max-SAT resolution step.

■

3.3.3 Weighted Max-SAT Resolution Calculus

A weighted version of Max-SAT resolution was introduced in [LH05a; BLM07; LHG08] and shown sound and complete for Max-SAT. We provide a formal definition of this rule below in compacted form [LHG08]. Note that this rule is often naturally augmented with the fold and unfold rules in Definition 3.13, in which case the premise clauses can be considered of equal weight w.l.o.g. In [LR20a], it was shown that augmenting MaxRes with the extension rule defined below produces a system stronger than MaxRes and at least as strong as dual rail. Note the particularity of the extension rule in allowing negative weights. A similar rule, called virtual, was introduced in [LR20b]. The system composed of Max-SAT resolution, split and virtual, called ResSV, was shown stronger than ResS (i.e., MaxRes augmented with split) and was also proven at least as strong as dual rail.

Definition 3.27 (Weighted Max-SAT Resolution [LH05a; BLM07; LHG08])**.** *The weighted Max-SAT resolution rule is defined as follows:*

$$\frac{(x \vee A, w_1) \qquad (\overline{x} \vee B, w_2)}{\begin{array}{c}(A \vee B, min(w_1, w_2)) \\ (x \vee A, w_1 - min(w_1, w_2)) \\ (\overline{x} \vee B, w_2 - min(w_1, w_2)) \\ (x \vee A \vee \overline{B}, min(w_1, w_2)) \\ (\overline{x} \vee \overline{A} \vee B, min(w_1, w_2))\end{array}}$$

Definition 3.28 (Extension [LR20a])**.** *Given a clause C, a finite weight $w > 0$ and a variable x, the extension rule is defined as follows:*

$$\frac{}{\begin{array}{c}(C, -w) \\ (x \vee C, w) \\ (\overline{x} \vee C, w)\end{array}}$$

Definition 3.29 (Virtual [LR20b])**.** *Given a clause C, a finite weight $w > 0$, the virtual rule is defined as follows:*

$$\frac{}{(C, w) \ (C, -w)}$$

3.4 Conclusion

In this chapter, we introduced the Maximum Satisfiability (Max-SAT) problem and its variants. We reviewed the major methods for Max-SAT solving. In particular, we provided a detailed review on Branch and Bound (BnB) methods for Max-SAT which perform exhaustive search and rely on extensive inference, specifically during lower bound computation. We also included a comprehensive review on SAT-based approaches, which take advantage of the power of SAT oracles as decision engines, as well as other methods including incomplete ones. Then, we focused on proof theory for Max-SAT and particularly on the Max-SAT resolution rule as a sound and refutationally complete rule for Max-SAT extending the well-known SAT resolution rule. We further recalled the unique features of Max-SAT resolution as an inference rule performing a transfer of knowledge and we presented its weighted form. Our contributions in Chapter 5 specifically revolve around Max-SAT and aim to provide further understanding and insights on the extensive use of inference during lower bound estimation in BnB algorithms for Max-SAT. Our work in 6 also pertains to this problem although it also touches upon SAT as we mainly aim to bridge the gap between inference mechanisms dedicated to SAT and Max-SAT.

Contributions

4 Bandits for Adaptive Branching in SAT through Restarts

Contents

4.1 Introduction & Motivation

The advent of the CDCL framework and of the powerful techniques designed for SAT these last decades led to a huge breakthrough in terms of SAT solving. Indeed, modern solvers are now able to effortlessly solve instances with millions of variables while it was difficult to solve ones involving only hundreds of variables in the early 90s. However, as solvers became very competitive, it naturally ensues that improving their efficiency is becoming more and more difficult. In fact, quoting Audemard and Simon, we point out that "*improving SAT solvers is often a cruel world. To give an idea, improving a solver by solving at least ten more instances (on a fixed set of benchmarks of a competition) is generally showing a critical new feature. In general, the winner of a competition is decided based on a couple of additional solved benchmarks*" [AS12].

Our contributions in this chapter fall within this scope of improving the efficiency of modern SAT solvers. We particularly focus on an important component in CDCL solvers which is the branching heuristic used to pick the next variable to branch on in the search tree. We recall, as showcased in Section 2.2.1.3.2, that the Variable State Independent Decaying Sum (VSIDS) [Mos+01] has been the dominant heuristic since its introduction

two decades ago. Liang and al. also recently devised the new Conflict History-Based (CHB) branching heuristic [Lia+16a], and showed that it is competitive with VSIDS. In the last years, VSIDS and CHB have dominated the heuristics landscape as practically all the CDCL solvers presented in recent SAT competitions incorporate a variant of one of them.

Recent research has also shown the relevance of reinforcement learning in designing efficient search heuristics for combinatorial problems including CSP [Sch18; XY18; Wat+20; PW20; HT21; Kor+22] and, more marginally, SAT [Lia+16a; Lia+16b]. One of the main challenges is defining a heuristic which can have high performance on any considered instance. Indeed, it is well-known that a heuristic can perform outstandingly on a family of instances while failing drastically on another. To this end, we propose to augment the branching component in CDCL solvers with a Multi-Armed Bandit (MAB) framework which takes advantage of the restart mechanism to pick an adequate heuristic between VSIDS and CHB. Note that simple combinations of VSIDS and CHB have shown promising results in the MapleCOMSPS solver which has won several medals in the 2016 and 2017 SAT competitions. MapleCOMSPS switched from VSIDS to CHB after a set amount of time in its earlier version [Lia+16c] while it alternated between both heuristics by allocating the same duration of restarts to each one in [Lia+17]. Yet, we still lack a thorough analysis and comparison of such strategies in the state-of-the-art as well as a comparison with new promising methods based on reinforcement learning. Our MAB framework is devised to incorporate more adaptive strategies which enable to conduct inference based on the mean reward of each heuristic to select the most relevant one. In particular, the reward function is used to estimate the efficiency of each heuristic by relying on information acquired during the runs between restarts. We rely on the Upper Confidence Bound (UCB) strategies introduced in Section 2.4.2 to choose an arm at each restart. Such strategies are compared to simple random and static strategies and are shown to achieve considerable gain in terms of SAT solving.

This chapter is organized as follows. First, we start by presenting our MAB framework which relies on the restart mechanism to adaptively choose a relevant heuristic. Then, we describe different strategies that can be used to combine VSIDS and CHB through the restart mechanism in CDCL solvers including static and random strategies as well as the MAB strategies based on our framework. A thorough experimental evaluation and comparison of the different strategies is conducted in the following sections showing that the considered strategies, specifically UCB strategies used in the context of our MAB framework, can bring further gains to practical SAT solving. Finally, we mention that our contributions in this chapter have been published in [CHT21a; CHT21d].

4.2 Multi-Armed Bandit Framework for SAT

In order to use MAB strategies, we first introduce a MAB framework for adaptive branching in SAT through restarts. Let $A = \{h_1, \ldots, h_K\}$ be the set of arms for the MAB containing different candidate heuristics. The trials are the runs, i.e., executions, of the CDCL algorithm between restarts. The proposed framework selects a heuristic h_i where $i \in \{1 \ldots K\}$ at each restart of the backtracking algorithm. To choose an arm, MAB relies on a reward function

calculated during each run to estimate the performance of the chosen heuristic. The reward function plays an important role in the proposed framework and has a direct impact on its efficiency. We choose a reward function that estimates the ability of a heuristic to reach conflicts quickly and efficiently. If t denotes the current run, the reward of arm $h \in A$ is calculated as follows:

$$r_t(h) = \frac{log_2(decisions_t)}{decidedVars_t}$$

$decisions_t$ and $decidedVars_t$ respectively denote the number of decisions and the number of variables fixed by branching in the run t. Consequently, the earlier conflicts are encountered in the search tree and the fewer variables are instantiated, the greater the assigned reward value will be for the corresponding heuristic. This reward function is closely related to the explored sub-tree measure introduced in [PW20].

4.3 Strategies to Combine VSIDS and CHB Using Restarts

In this section, we describe different strategies which take advantage of the restart mechanism in SAT solvers to combine VSIDS and CHB. First, we describe simple strategies which are either random or static then we specify the reinforcement learning strategies that we will use in the context of the MAB framework.

Random Strategy (RD_R) This strategy randomly picks a heuristic among VSIDS and CHB at each restart with equal probabilities, i.e., each heuristic is assigned a probability of $\frac{1}{2}$. This strategy is denoted RD_R in contrast with RD_D which randomly picks a heuristic at each decision.

Single Switch Strategy (SS) This strategy switches from VSIDS to CHB after a set amount of time and was used in the 2016 version of MapleCOMSPS [Lia+16c]. We maintain the threshold time in which the heuristic is switched to $\frac{t}{2}$ where t is the timeout as in [Lia+16c].

Round-Robin Strategy (RR) This strategy alternates between VSIDS and CHB in the form of a round-robin. This is similar to the strategy used in the latest version of MapleCOMSPS [Lia+17]. However, since we want to consider strategies which are independent from the restart policy and which only focus on choosing the heuristics, we do not assign equal amounts of restart duration (in terms of number of conflicts) to each heuristic and, instead, let the duration of restarts augment naturally with respect to the restart policy of the solver.

Upper Confidence Bound Strategies ($UCB1$ & $MOSS$) in the context of our MAB framework, we consider the deterministic UCB family of strategies, namely the UCB1

and MOSS strategies presented in 2.4.2. Although these strategies have not been experimented upon in the context of SAT, they have shown promising results in the context of combinatorial problem solving and particularly the CSP problem [XY18; Wat+20; PW20; Kor+22].

4.4 Experimental Protocol

Before presenting the results of our experimental evaluation, we detail in this section our experimental protocol. We consider the benchmarks from the Main Track of the last three SAT Competitions/Races, totalling to 1,200 instances. For our experiments, we use the state-of-the-art solver Kissat [Bie+20] which won first place in the main track of the SAT Competition 2020. Note that this solver is a condensed and improved reimplementation of the reference and competitive solver CaDiCaL [Bie17; Bie+20] in C. Data provided by Bierre and Marjin [1] show that Kissat is highly competitive and outperforms all-time winners of SAT competitions/Races particularly on the 2020 and 2019 Benchmarks. Kissat alternates between stable and non-stable phases as is the case in Cadical [Bie17], renamed to stable mode and focused mode in [Bie+20]. VSIDS is used in stable phases which mainly target satisfiable instances. During non-stable phases targeting unsatisfiable instances, the solver uses the Variable Move-To-Front (VMTF) heuristic [Rya04; BF15] [2], in which analyzed variables are moved to the front of the decision queue. It is important to note that the only modified components of the solver are the decision component and the restart component, i.e., all the other components as well as the default parameters of the solver are left untouched. Even the changes to the restart component are as minimal as possible, i.e., we maintain the phase alternation mechanism and the restart policies set for each mode as described in [Bie+20]. Furthermore, we maintain the VSIDS variant already implemented in Kissat, i.e., EVSIDS [Bie08a; BF15]. Therefore, in the experimental evaluation, VSIDS corresponds to default Kissat. Moreover, we augment the solver with the heuristic CHB as specified in [Lia+16a] except that we update the scores of the variables in the last decision level after BCP. In addition, we have implemented the MAB framework specified in Section 4.2 with $A = \{VSIDS, CHB\}$. The rewards for UCB1 and MOSS are both initialized by launching each heuristic once during the first restarts. Finally, The experiments are performed on Dell PowerEdge M620 servers with Intel Xeon Silver E5-2609 processors under Ubuntu 18.04 with a timeout of 5,000 s for each instance.

4.5 Decisions vs Restarts

First, we would like to emphasize that taking advantage of the restart mechanism to combine VSIDS and CHB was not an arbitrary choice. Indeed, we conducted an experiment to help us choose the appropriate level, i.e., decisions or restarts, to combine VSIDS and CHB. To this end, we implemented and tested the two random strategies RD_D and RD_R which

[1] Data available on `http://fmv.jku.at/kissat/`
[2] refer to Section 2.2.1.3.2

		VSIDS	CHB	RD_D	RD_R	SS	RR	UCB1	MOSS	VBS
Competition 2018 (400 instances)	SAT	160	159	160	164	163	165	167	**168**	169
	UNSAT	111	109	109	110	**113**	110	110	110	113
	TOTAL	271	268	268	274	276	275	277	**278**	282
Race 2019 (400 instances)	SAT	158	149	155	158	154	**162**	161	**162**	162
	UNSAT	**97**	95	95	96	96	96	96	**97**	99
	TOTAL	255	244	250	254	250	258	257	**259**	261
Competition 2020 (400 instances)	SAT	131	146	146	151	147	152	154	**156**	157
	UNSAT	121	119	117	120	118	120	120	**122**	123
	TOTAL	252	265	263	271	265	272	274	**278**	280
TOTAL (1,200 instances)	SAT	449	454	461	473	464	479	482	**486**	488
	UNSAT	**329**	323	321	326	327	326	326	**329**	335
	TOTAL	778	777	782	799	791	805	808	**815**	823

Table 4.1: Comparison between VSIDS, CHB, the different strategies and the VBS (over VSIDS and CHB) in terms of the number of solved instances in Kissat. For each row, the best results without considering the VBS are written in bold.

randomly choose a heuristic among VSIDS and CHB respectively in each decision and in each restart. The average results (over 10 runs with different seeds) of RD_D and RD_R on the whole benchmark are reported in Table 4.1 and indicate that RD_R outperforms RD_D with a gain of more than 2% in terms of solved instances and 3.5% in terms of solving time with a penalty of 10,000 s for unsolved instances. This is not surprising as the structures needed for VSIDS and CHB need to be maintained and updated simultaneously which can be quite costly. On the other hand, they are used independently in RD_R during each restart, i.e., only the chosen heuristic is used and its structures updated during the restart. Furthermore, combining both heuristics at the decision level can cause interference and may not allow each heuristic to conduct robust learning since they are being constantly interchanged. Surprisingly, both versions are competitive with CHB and VSIDS. In particular, RD_R outperforms them and solves, on average, 21 additional instances (+ 2.7%) compared to the best heuristic. This is due to randomization and diversification which help to avoid heavy tail phenomena in SAT and which can therefore improve the performance of SAT solvers [HG95; Gom+00].

4.6 Comparison of Strategies

In this section, we evaluate and compare the different strategies aiming to combine VSIDS and CHB through restarts in terms of solved instances and solving time. We also analyze the results achieved by these strategies with respect to instance families within the benchmark.

4.6.1 Number of Solved Instances

In Table 4.1, we present the results in terms of solved instances for CHB and VSIDS as standalone heuristics and for the different strategies presented in Section 4.3. We also include the results of the Virtual Best Solver (VBS) over VSIDS and CHB [3]. The results clearly indicate that MOSS outperforms VSIDS and CHB as well as all the other strategies. Indeed, MOSS manages to solve 37 additional instances in total (+4.8%) compared to the best heuristic (among VSIDS and CHB). The UCB1 (resp. RR) strategy is also competitive and manages to solve 30 (resp. 27) additional instances in total which corresponds to an increase of 3.9% (resp. 3.5%) in terms of solved instances compared to the best heuristic. The strategies UCB1 and RR remain comparable with a difference of 3 instances in favor of UCB1. SS also outperforms VSIDS and CHB although to a lesser degree as it solves 13 additional instances only which is worse than RD_R. If we focus on the individual yearly benchmarks, we observe that although the overall results obtained by VSIDS and CHB are comparable, they have different behaviours on each benchmark and yet MOSS, UCB1 and RR manage to capture the behaviour of the best heuristic and even outperform it on each individual benchmark. In particular, MOSS maintains its top rank on the individual benchmarks with an average of 8 (resp. 17) additional instances for each one compared to the best (resp. worst) heuristic. Moreover, the results achieved by MOSS are very close to the VBS. Indeed it achieves 99% (resp. 99.6%) of the performance of the VBS on the whole benchmark in terms of the number of solved instances (resp. satisfiable instances) while the best heuristic does not exceed 95% (resp. 93%).

However, it is important to note that the gain is mainly in satisfiable instances whereas, for unsatisfiable instances, all the strategies (except RD_D) remain comparable to both heuristics and slightly outperform CHB but not VSIDS. Nevertheless, they remain competitive with VSIDS and particularly MOSS which solves the same number of unsatisfiable instances as VSIDS. This shows that MOSS is a robust strategy as it is able to improve the performance globally and on each individual benchmark without decreasing it for unsatisfiable instances. Note that the observed behaviour of these different strategies on unsatisfiable instances may be due to different factors. First, the results in terms of unsatisfiable instances seem very homogeneous for each year and are very close to the results obtained by the VBS as both heuristics (resp. the best heuristic) achieve more than 96% (resp. 98%) of its performance in terms of the number of unsatisfiable instances. Since our motivation is to bridge the gap between the heuristics and the VBS with these strategies, it is expected that this would be very difficult for unsatisfiable instances, for which the gap is very small already. It is also very difficult to simultaneously improve the performance on both satisfiable and unsatisfiable instances. For instance, SS which seems to work better for unsatisfiable instances especially in terms of solving time (as will be showcased in the next section) fails on satisfiable instances compared to the three top strategies. Another possible factor for this behaviour is Kissat's restarting policy which alternates between the stable mode and focused mode [Bie+20]. The heuristics VSIDS and CHB are only used in

[3] If an instance is solved by one heuristic, we report its results in the VBS. If both heuristics manage to solve an instance, we report the best result in the VBS. The VBS over VSIDS and CHB thus gives an estimate on the best performance that can be expected from a solver employing both heuristics.

the stable mode while the focused mode targets unsatisfiable instances. This may also help to explain the homogeneity of the results obtained by the solver for unsatisfiable instances with respect to the different heuristics and strategies.

4.6.2 Solving Time

In this section, we want to evaluate the different strategies in terms of solving time. In Figure 4.1, we represent the number of solved instances as a function of the execution time for VSIDS, CHB, the static and MAB strategies and the VBS on the whole benchmark. One would think that MAB based strategies in this regard would be worse than the considered heuristics and/or other strategies as UCB1 and MOSS need to conduct continuous exploration in order to ensure the selection of the most adequate arm. This does not seem to be the case. In fact, conducting exploitation with the best arm and alternating the heuristics seems to offset this disadvantage. We observe that MOSS is the best strategy as it achieves 6.1% gain in terms of solving time on the whole benchmark compared to the best heuristic if we give a penalty 10,000 s to unsolved instances while UCB1, RR and SS respectively achieve a gain of 5.7%, 5.2% and 1.7%. This gain is substantial especially considering that we are working on the solver Kissat which won the SAT competition 2020 with a remarkable performance.

We represent in Figure 4.2 the number of solved satisfiable and unsatisfiable instances separately as a function of the execution time for VSIDS, CHB, the static and MAB strategies and the VBS on the whole benchmark. Notice how the gap between MOSS and the VBS (and even UCB1 and RR) narrows if we consider the satisfiable instances only. On the other hand, these top three strategies present a small gap in terms of solving time for unsatisfiable instances compared to the best heuristic, i.e., VSIDS, while remaining comparable to CHB. In particular, MOSS shows better results with respect to VSIDS and SS for instances whose solving time exceeds 3,000 s. Surprisingly, although SS seems to be the worst strategy overall and remains globally comparable to VSIDS and CHB while achieving a slight gain in solving time especially on instances whose solving time exceeds 4,000 s, it achieves the best results in terms of solving time for unsatisfiable instances and is comparable to VSIDS and the VBS in this regard. On the other hand, RR and UCB1 achieve substantial gain while remaining comparable to each other and with results slightly in favor of UCB1. To provide more detailed results, we represent in Figures 4.3, 4.4 and 4.5 the runtime comparison per instance with VSIDS, CHB and the VBS respectively for the top three best strategies, i.e., MOSS, UCB1 and RR. These figures confirm the trends that we observed above. More interesting, we can note that, for a noticeable number of instances, MOSS, UCB1 or RR lead to a more efficient solving than the VBS. In Figure 4.6, we represent the runtime comparison per instance betwenn MOSS, UCB1 and RR. These figures show that MOSS performs better than UCB1 and RR. Surprisingly, MOSS's results are closer to RR than UCB1. However, we will show in Section 4.7.1 that this is consistent with the observed behavior of MOSS.

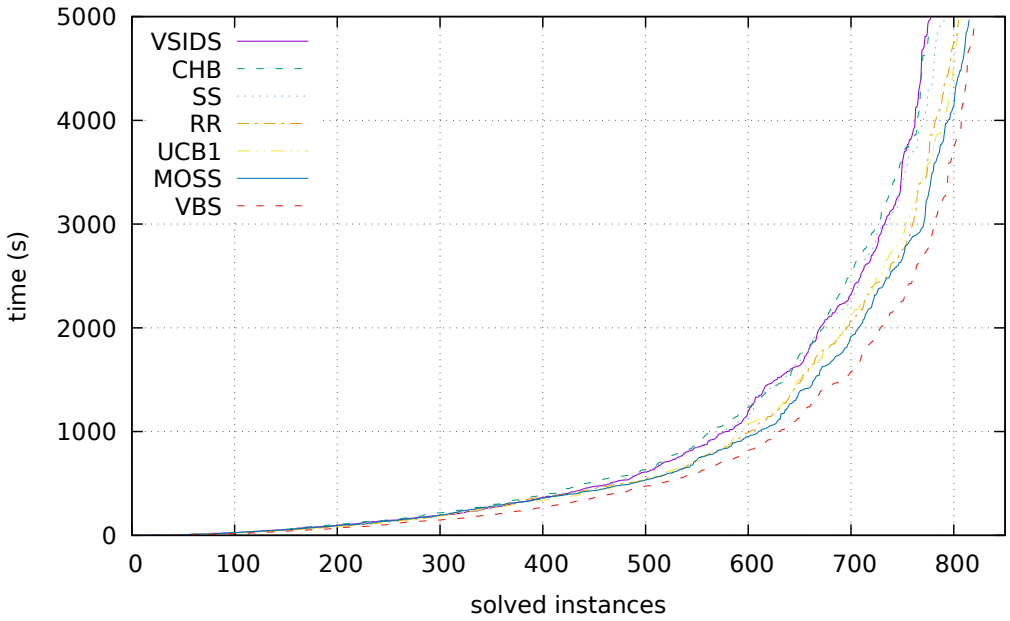
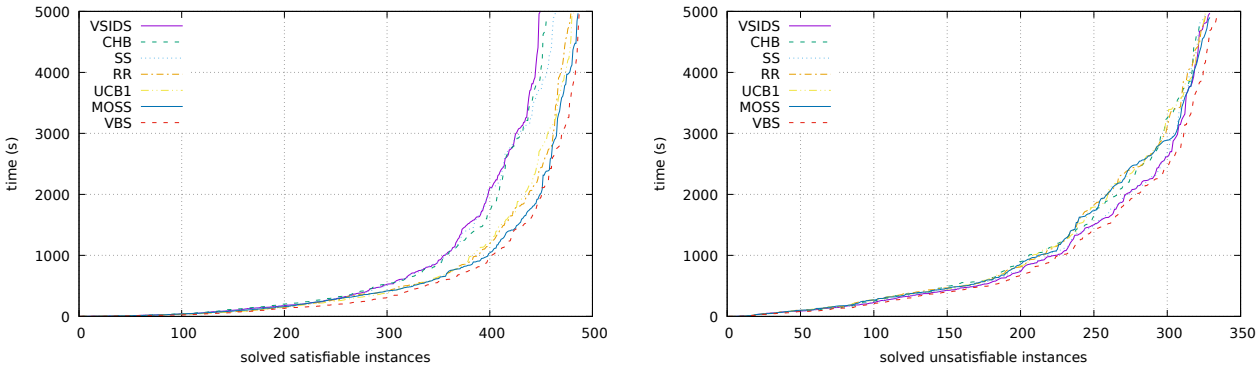
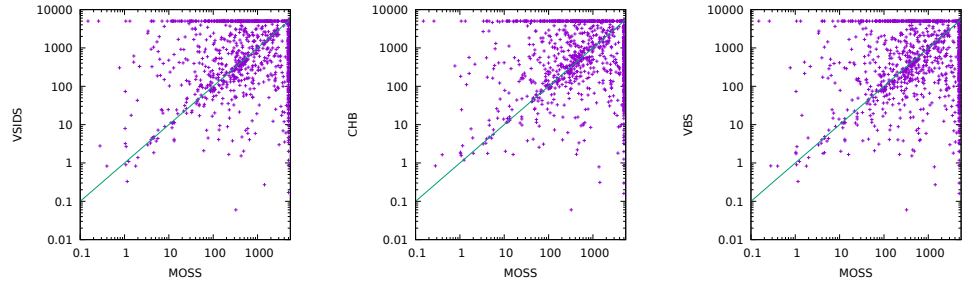


Figure 4.1: Number of solved instances as a function of execution time for VSIDS, CHB, static and MAB strategies and the VBS w.r.t the whole benchmark.



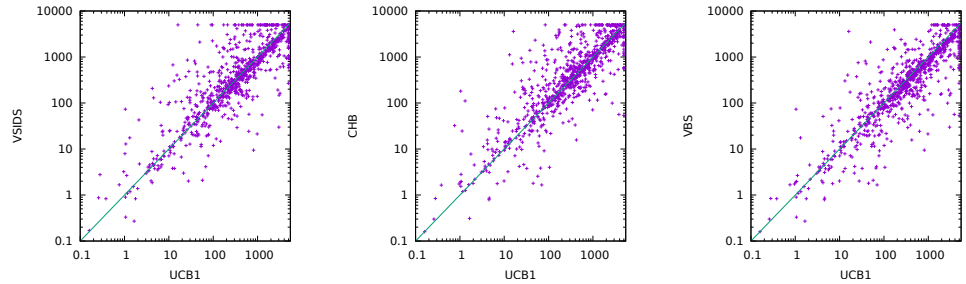
(a) (b)

Figure 4.2: Number of solved satisfiable (a) and unsatisfiable (b) instances as a function of execution time for VSIDS, CHB, static and MAB strategies and the VBS w.r.t the whole benchmark.



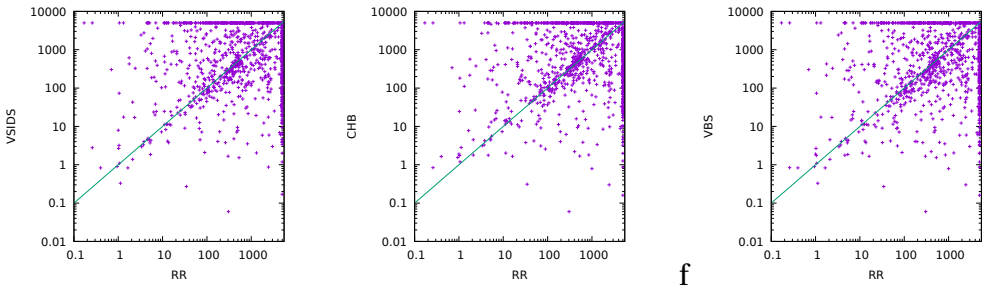
(a) (b) (c)

Figure 4.3: Runtime comparison (in seconds) of MOSS w.r.t. VSIDS (a), CHB (b) and VBS (c) in logarithmic scale.



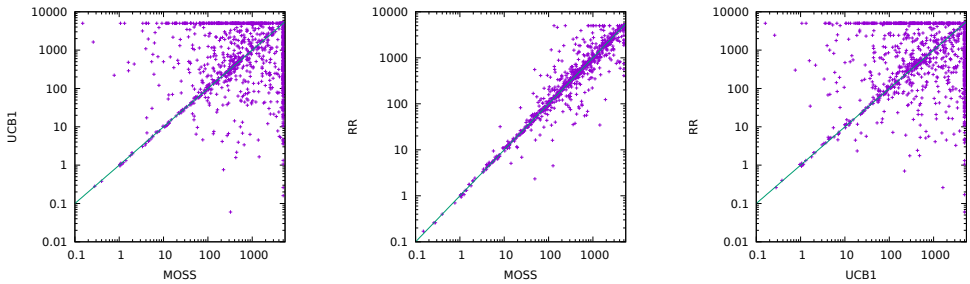
(a) (b) (c)

Figure 4.4: Runtime comparison (in seconds) of UCB1 w.r.t. VSIDS (a), CHB (b) and VBS (c) in logarithmic scale.



(a) (b) f (c)

Figure 4.5: Runtime comparison (in seconds) of RR w.r.t. VSIDS (a), CHB (b) and VBS (c) in logarithmic scale.



(a) (b) (c)

Figure 4.6: Runtime comparison (in seconds) of MOSS w.r.t. UCB1 (a) and RR (b) and of UCB1 w.r.t. RR (c) in logarithmic scale.

4.6.3 Instance Families

In order to provide a more thorough analysis, we describe in Tables 4.2 and 4.3 the results obtained by VSIDS, CHB, static and MAB strategies and the VBS on instance families within the benchmark [HJS+18; HJS19; Bal+20]. The best strategy, i.e., MOSS, manages to rank first in 9 different families over 39 in total (23%), e.g., it places fist for `Antibandwidth`, `Bitcoin` and `Stedman Triples`. Interestingly, this strategy achieves remarkable results, which are better than those of the VBS over VSIDS and CHB, for certain families such as `Logical cryptanalysis`, `RPHP` and `Station Repacking`. SS also achieves the top performance on several different families such as `Factoring`, `Scrambled` and `SHA-1 Pre-image Attack`. More precisely, SS also manages to rank on top for 9 different families which shows the interest of this strategy even though it ranks last overall compared to RR, UCB1 and MOSS. As for UCB1, it achieves top rank in 6 different families. In particular, its performance on the families `Hgen`, `CNP` and `Keystream Generator Cryptanalysis` is noteworthy since it manages to outperform the VBS. On the other hand, RR ranks top in only 4 instance families but this does not necessarily reflect its overall performance since it falls slightly behind the top ranked heuristic/strategy in other families, yet this is clearly another point in favor of UCB1 as a comparable strategy. Finally, VSIDS and CHB are ranked first in several families which shows that these heuristics remain robust as standalone heuristics.

4.7 Further Analysis of MAB Strategies

In this section, we cconduct a more through analysis on the behavior of UCB strategies used in the context of the MAB framework. We also include a discussion on the MAB framework and specifically the relevance of some choices that we made within this context.

4.7.1 MAB Behaviour

In this section, we focus on the behaviour of MAB strategies and particularly the use of arms. In Figures 4.7 and 4.8, we represent the percentage of use, i.e., percentage of restarts where each arm gets chosen respectively by UCB1 and MOSS. We observe that both strategies alternate between the heuristics but the percentages are mainly within the interval $[40\%, 60\%]$ and are often close to 50%. MOSS seems to choose in a more balanced way between VSIDS and CHB in comparison to UCB1 which introduces more variations in its choices. This behaviour is consistent with the observations made in Section 4.6.1 concerning Figure 4.6. The fact that the percentages are mostly within a tight interval is not surprising considering that the number of stable restarts in Kissat, during which heuristics are used, is usually very low. To give an idea, the average number of stable restarts performed by Kissat for instances solved with MOSS (resp. UCB1) is 765 (resp. 771) while the median value is much lower and amounts to 313 (resp. 338). Therefore, the obtained percentages seem adequate especially taking into account that these strategies need to achieve a good trade-off between exploration and exploitation. Notice the consecutive dents and bumps in Figures 4.7 and 4.8 which correspond to an homogeneous behaviour within the same instance family in the benchmark. It is important to note that, although the

Family		VSIDS		CHB		SS		RR		UCB1		MOSS		VBS	
name	**#I**	**#I**	**time**	**#I**	**time**	**#I**	**time**	**#I**	**time**	**#I**	**time**	**#I**	**time**	**#I**	**time**
Antibandwidth	14	2	1,010	7	9,804	7	21,381	8	11,469	9	15,651	**9**	**14,370**	7	9,628
Almost Perfect Non-Linear S-box Finder	20	**11**	**15,324**	7	14,386	11	18,815	10	18,974	11	19,659	11	15,969	12	16,138
Arithmetic Verification	38	13	11,010	**14**	**12,268**	8	6,657	9	12,160	13	13,349	9	11,811	14	10,801
Baseball-lineup	13	12	3,317	12	2,949	12	3,192	12	2,645	12	2,947	**12**	**2,540**	12	2,906
Bitcoin	17	8	1,972	7	479	8	2,001	8	1,907	8	2,199	**8**	**1,901**	8	1,784
Coloring	14	6	7,501	5	7,327	**7**	**12,097**	5	3,101	5	2,753	6	7,556	6	5,876
Core-based	14	13	8,438	13	10,860	13	8,737	13	7,431	**14**	**14,165**	14	14,861	13	7,239
Course Scheduling	20	14	14,439	14	15,363	13	11,205	14	11,804	**15**	**9,323**	15	10,801	14	9,654
Cover	13	**4**	**9**	4	10	4	9	4	10	4	10	4	11	4	9
Chromatic Number (CNP)	20	20	1,708	20	1,972	20	1,743	20	1,402	**20**	**1,180**	20	1,415	20	1,194
Divide and Unique Inverse	20	**16**	**18,456**	16	22,537	16	19,615	16	20,864	16	23,543	16	20,931	16	18,109
Discrete-Logarithm	7	4	3,640	4	7,623	**4**	**3,344**	4	4,718	4	4,894	4	5,883	4	3,627
Edge-Matching Puzzle †	14	3	4,476	3	6,201	2	1,430	**4**	**5,546**	4	8,674	4	6,615	4	8,823
Factoring †	32	30	17,224	27	12,497	**30**	**16,554**	27	10,297	28	20,528	28	20,106	30	12,546
Floating-Point Program Verification	15	12	972	**12**	**874**	12	1,024	12	1,050	12	1,076	12	991	12	775
Grand Tour Puzzle	19	9	1,834	9	2,037	**9**	**1,771**	9	2,042	9	2,061	9	2,153	9	1,783
Hard 3-SAT	20	18	5,486	19	8,888	18	6,424	**19**	**3,654**	18	3,643	19	4,042	19	7,238
Hgen	13	12	3,168	12	2,423	12	3,134	12	1,783	**12**	**783**	12	2,590	12	2,365
Influence Maximization	14	12	9,617	**12**	**7,424**	12	9,472	12	10,037	12	9,989	12	10,152	12	6,865
Kakuro Puzzle	14	**12**	**15,705**	11	13,942	11	10,312	12	16,365	12	16,269	12	16,216	12	14,582

Table 4.2: Comparison between VSIDS, CHB, static and MAB strategies and the VBS (over VSIDS and CHB) in terms of the number of solved instances (#I) and cumulative solving time (for solved instances in seconds) in Kissat for instance families in the benchmark. The results of families marked with † are joint from two different yearly benchmarks. For each row, the best results without considering the VBS are written in bold, breaking ties with milliseconds if necessary.

Family		VSIDS		CHB		SS		RR		UCB1		MOSS		VBS	
name	**#I**	**#I**	**time**	**#I**	**time**	**#I**	**time**	**#I**	**time**	**#I**	**time**	**#I**	**time**	**#I**	**time**
k-Colorability	15	5	7,923	5	6,528	**6**	**12,338**	5	5,167	5	6,998	5	5,540	6	10,660
Keystream Generator Cryptanalysis	18	18	14,494	14	15,104	18	14,731	18	19,433	**18**	**13,118**	18	16,828	18	13,240
Lam-Discrete-Geometry	9	**7**	**4,756**	7	7,106	7	4,899	7	7,147	7	6,856	7	6,953	7	4,680
Logical Cryptanalysis	20	20	5,606	20	10,476	20	6,748	20	5,518	20	4,241	**20**	**4,208**	20	4,946
Polynomial Multiplication †	27	20	28,884	21	27,880	21	33,016	21	27,107	20	23,653	**22**	**30,535**	25	41,331
Population Safety	15	13	2,188	**14**	**1,991**	13	2,423	12	1,624	13	3,148	13	2,417	14	1,809
Preimage	11	6	11,865	4	7,998	**7**	**13,070**	6	16,201	5	9,255	5	5,852	8	15,435
Relativized Pigeonhole Principle (RPHP)	20	11	14,890	10	11,344	11	15,229	10	9,869	11	14,065	**11**	**13,967**	11	14,890
Reversing Elementary Cellular Automata	11	11	4,046	11	4,065	**11**	**3,923**	11	4,738	11	5,151	11	4,476	11	3,664
Scrambled	20	19	7,022	18	9,278	**20**	**11,441**	19	8,114	19	14,519	20	15,141	20	6,089
SHA-1 Pre-image Attack	20	20	14,509	20	23,429	**20**	**14,085**	19	21,975	20	25,206	20	20,255	20	12,214
Social Golfer	14	2	3,008	1	3,791	1	410	2	1,202	**3**	**6,046**	2	1,530	2	3,008
Software Bounded Model Checking	19	**18**	**7,082**	18	8,959	18	7,219	18	7,966	18	8,308	18	8,435	18	6,969
Station Repacking	12	6	15,286	12	10,656	11	29,669	12	13,752	12	8,766	**12**	**6,855**	12	10,656
Stedman Triples †	27	10	8,766	11	11,508	11	11,394	12	8,215	12	7,399	**13**	**12,329**	11	6,947
SV Competition	18	18	7,522	17	3,350	17	4,227	**18**	**7,251**	18	7,935	18	7,829	18	5,577
Timetable †	26	1	1565	10	5082	10	28,417	11	5,607	11	6,247	**11**	**5,157**	10	5,082
Tree Decomposition	20	11	12,049	10	6,870	10	7,947	**11**	**4,700**	10	3,965	11	5,674	11	7,874
Vlsat	14	3	103	7	4,457	4	3,404	7	529	**7**	**500**	7	547	7	3,934

Table 4.3: Comparison between VSIDS, CHB, static and MAB strategies and the VBS (over VSIDS and CHB) in terms of the number of solved instances (#I) and cumulative solving time (for solved instances in seconds) in Kissat for some instance families in the benchmark (Table 4.2 continued). The results of families marked with † are joint from two different yearly benchmarks. For each row, the best results without considering the VBS are written in bold, breaking ties with milliseconds if necessary.

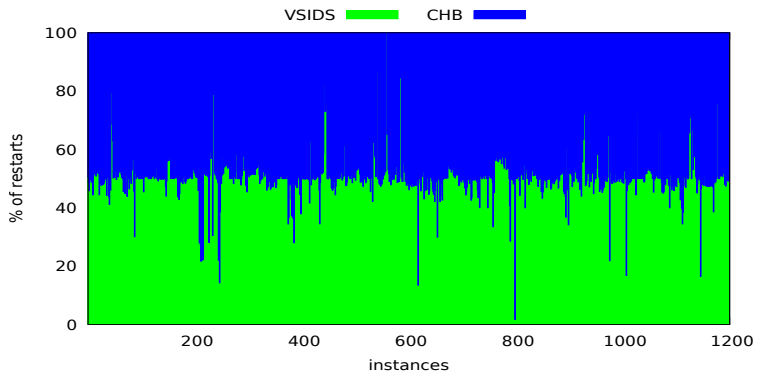


Figure 4.7: Percentages of use of each arm in UCB1 w.r.t the whole benchmark. The instances are reported consecutively for each yearly benchmark (from 2018 to 2020) and are alphabetically ordered. For unsolved instances, the percentages of use at the timeout are provided.

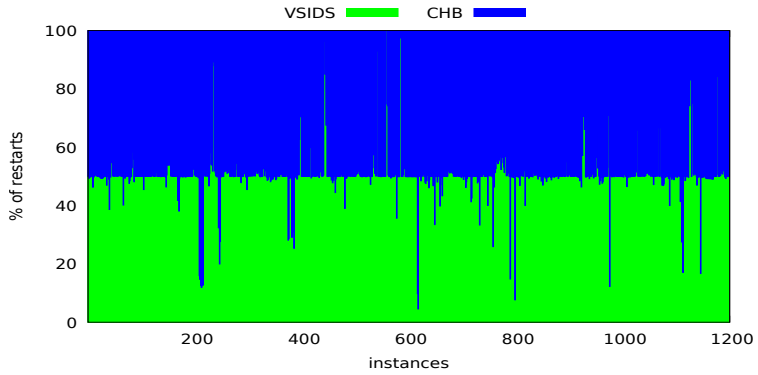


Figure 4.8: Percentages of use of each arm in MOSS w.r.t the whole benchmark. The instances are reported consecutively for each yearly benchmark (from 2018 to 2020) and are alphabetically ordered. For unsolved instances, the percentages of use at the timeout are provided.

behaviour of MAB strategies may seem close to RR, this is not exactly the case. Indeed, these strategies rely on the computed reward to choose the most relevant arm during exploitation and especially when there is a large gap between the performance of the heuristics, whereas RR is a static strategy and cannot adapt its choices. This helps to explain the better results of the MAB strategies not only in terms of solved instances but also in terms of solving time and particularly in the case of MOSS. In fact, the remarkable performance of MOSS is also due to the fact that it takes into account the number of arms and has better regret than UCB1 although the latter also remains competitive in practice as shown in our results.

4.7.2 On MAB strategies and Branching Heuristics

In this section, we discuss the relevance of choosing Upper Confidence Bound strategies in the Multi-Armed Bandit framework and VSIDS and CHB as candidate heuristics. As

mentioned in Section 2.4.2, many strategies were divised and theoretically studied in the context of MAB and can therefore be used in our framework. For instance, we can mention the two well-known strategies ϵ-greedy [SB98] and EXP3 [Aue+02] which were presented in Section 2.4.2. However, these strategies are not deterministic, i.e., there is a factor of uncertainty or probability. Therefore, unlike UCB strategies, they cannot always guarantee top performance and may produce different results on the same benchmark. Furthermore, UCB strategies were shown relevant and more efficient for similar MAB frameworks in the context of CSP [XY18; Wat+20; PW20; Kor+22]. This remains true in Kissat as we similarly observed through experimentation that ϵ-greedy and EXP3 perform poorly compared to UCB strategies and remain comparable to VSIDS and CHB.

In addition, notice that the MAB framework enables the use of several heuristics. In fact, one would argue that adding more heuristics may enable to reach more families and instances through diversification. However, recall that modern SAT solvers, and in particular Kissat, are highly competitive and rely on powerful heuristics to achieve impressive results. A bad heuristic or tuning of the parameters (e.g., the restart policy settings) can greatly deteriorate the performance of a solver. Furthermore, practically all heuristics used in modern SAT solvers are variants of VSIDS, which has been the dominant heuristic since its introduction in 2001 [Mos+01]. Only recently CHB has been introduced and shown competitive with VSIDS [Lia+16a]. The results reported in Table 4.1 also show that CHB can reach new instances (the VBS achieves a gain of more than 5.8% in terms of solved instances) while remaining competitive and comparable overall with respect to VSIDS in the context of a highly competitive solver such as Kissat.

4.8 Kissat_MAB at the SAT Competitions

We submitted the solver Kissat augmented with our MAB framework relying on the UCB1 strategy to the SAT competition 2021 under the name Kissat_MAB [4] [CHT21b]. This solver won the Main Track of the competition and managed to solve 296 instances over 400 with a gap of 8 instances compared to the second ranked solver. The summary of the Main Track results are reported in Figure 4.9. Kissat_MAB also placed first in the Main SAT and NoLimits tracks. Compared to default Kissat, which also participated in the competition under the name Kissat_sc2021_default with several new improvements over its last version [BFH21], Kissat_MAB achieves better results with 9 (resp. 11) additional solved (resp. satisfiable) instances. Furthermore, Kissat_MAB remains highly competitive on unsatisfiable instances and comparable to default Kissat as it managed to solve 148, only 2 instances less than Kissat_sc2021_default. We also submitted the solver Kissat augmented with both UCB strategies, i.e., UCB1 and MOSS, to the SAT competition 2022 under the names Kissat_MAB_UCB and Kissat_MAB_MOSS [5] [CHT22a]. The solvers placed respectively fifth and fourth in the Main sequential track solving 287 and 288 instances over 400. Note that the first three solvers in the track are all based on our Kissat_MAB solver [Bal+22]. The results of the SAT competitions thus seem to corroborate our experimental study and to confirm

[4] Results and source code available on https://satcompetition.github.io/2021/.
[5] Results and source code available on https://satcompetition.github.io/2022/.

the relevance of combining VSIDS and CHB using restarts in improving the performance of highly competitive SAT solvers.

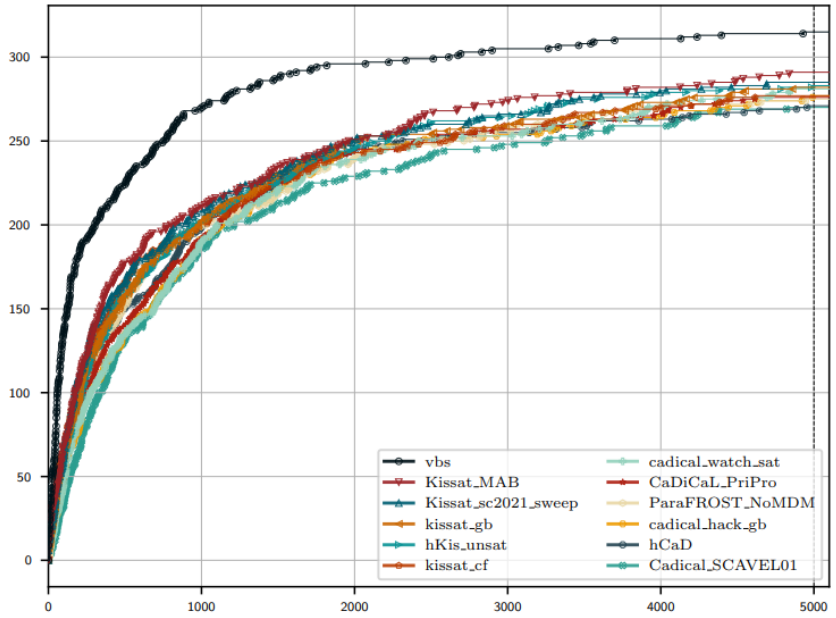


Figure 4.9: Main tack results of the 2021 SAT competition [Bal+21]

4.9 Conclusion & Future Work

In this chapter, we evaluated different strategies which take advantage of the restart mechanism to combine two state of the art heuristics, namely VSIDS and CHB. In particular, we introduced a MAB framework for SAT relying on two well-known Upper Confidence Bound strategies, called UCB1 and MOSS. These strategies rely on a reward function which evaluates the capacity of the heuristics to reach conflicts quickly and efficiently. Our experimental evaluation shows that VSIDS and CHB are compatible since their combination through different strategies taking advantage of the restart mechanism is able to substantially increase the performance of the competitive solver Kissat. In particular, the MOSS and UCB1 strategies outperform VSIDS and CHB as well as the other considered strategies. Overall, the considered strategies achieve substantial gain in terms of solved instances, mainly satisfiable ones, and in terms of solving time. Moreover, these strategies achieve results which are very close to the VBS over VSIDS and CHB. Our solver Kissat_MAB won several medals in the two last SAT competition including the gold medal the Main track of the SAT competition 2021 thus showing the relevance of combining VISDS and CHB using restarts and its ability to improve the performance of highly competitive SAT solvers.

Our work in terms of incorporating reinforcement learning techniques into combinatorial problem solving also extends to the CSP problem. In [CHT20; CHT21c], we devised a MAB framework, coupled with a training phase, to select an appropriate value of the

step parameter in the CHS heuristic [6] [HT21] during the restarts performed by the search algorithm. Each arm represents a different value for this parameter and is rewarded by its ability to improve the search. A training phase is introduced earlier in the search to help MAB choose a relevant arm. The experimental evaluation shows that this approach leads to significant improvements over plain CHS and other state-of-the-art heuristics in CSP solving.

As future work, it would be interesting to refine the reward function used in MAB strategies by relying on a combination of different criteria [Chu+11a] so as to improve the MAB framework especially with respect to unsatisfiable instances. Furthermore, it would be relevant to use such strategies to improve other components in modern SAT solvers. For instance, the attempt in [Li+22b] uses our MAB framework to perform adaptive restarts by choosing a relevant restart strategy at each trial. A similar mechanism can be devised for other components such as clause deletion or inference techniques. Finally, since Max-SAT solvers rely heavily on SAT oracles as decision engines, our work can naturally enhance Max-SAT solving specifically in the context of SAT-based solvers and, more marginally, in the context of BnB solvers [7]. However, it would be relevant to devise dedicated adaptive mechanisms relying on similar MAB frameworks in the context Max-SAT and, particularly, BnB solvers such as MaxCDCL [Li+21a; Li+22a].

.

[6] We recall that this heuristic is an adaptation of the CHB heuristic in the context of CSP and similarly includes a step parameter as explained for CHB in Section 2.2.1.3.2.

[7] refer to Section 3.2

5 Understanding Inference in Max-SAT BnB

Contents

5.1 Introduction & Motivation

Brand and Bound (BnB) based approaches construct a search tree and compute, at each node, the Lower Bound (LB) by counting the disjoint Inconsistent Subsets (ISs) of the formula using Simulated Unit Propagation (SUP) [LMP05]. When an IS is found, it is either temporarily deleted or transformed by Max-SAT resolution to ensure that it will be counted only once. However, learning Max-SAT resolution transformations, i.e., memorizing them in the current subtree, may negatively affect the quality of the lower bound estimation. Therefore, state-of-the-art solvers learn transformations selectively mainly in the form of patterns [LMP07; AH14d]. The most significant feature of these patterns is reducing the size of the formula or producing unit clauses whose propagation through SUP may lead to the detection of more ISs.

The empirical study of the Unit Clause Subset (UCS) patterns in [AH14d] led to the first observations on the relation between Max-SAT resolution transformations and the efficiency of the SUP mechanism. It was particularly observed that in some cases the information which can be used by SUP in the original formula are fragmented into several clauses after the transformation. This observation was later formalized through the the notion of UP-resilience introduced in [AH15b]. This property aims to characterize the transformations which are not affected by fragmentation and, more generally, to measure the impact of the transformations on the SUP mechanism. The study of the major learning

mechanisms through this property helped to explain, from a theoretical point of view, the empirical results of Max-SAT BnB obtained in the last decade.

In this chapter, we further investigate the power of inference in the context of Max-SAT BnB through the UP-resilience property. We conduct a more through analysis of this property which helps to shed the light on more recent results in the literature. In particular, we further study the relation between UP-resilience and the orders of application of Max-SAT resolution when transforming ISs. Relying on our observations on the direct impact of application orders on the UP-resilience of literals appearing in ISs, we conduct a theoretical study on the recent UCS patterns and their relation with UP-resilience. We prove that binary UCSs are UP-resilient with respect to specific orders and we generalize this result on UCSs where only one clause of any size is involved in the conflict. Furthermore, we explain how our results can help extend the current learning schemes used in Max-SAT BnB by studying their limits.

This chapter is organized as follows. We first state some known results to which we provide simpler or new formal proofs including results on the relation between UP-resilience and many components and mechanisms in Max-SAT BnB such as IS detection, application orders during IS transformation and the traditional patterns in the literature. Then, we further investigate the recently introduced and their relation with UP-resilience. We also show the limits of the current mechanisms thus providing insights on how to extend them through the UP-resilience property. Finally, we mention that our contributions in this chapter have been published in [CHA20; CHA21].

5.2 UP-Resilience and IS detection

One of the most interesting properties of UP-resilient transformations is their ability to maintain the propagations which are not necessary anymore to the detection of an IS. As explained in Section 3.2.1.5, if a transformation is UP-resilient for a literal l, then l can be propagated in the transformed formula when the literals of each possible neighborhood, not containing the empty clause, are set to True. This is extended to sets of literals as stated in Proposition 3.1, i.e., if a transformation is UP-resilient for a set of literals L, then the literals of L can be propagated in the transformed formula with respect to the possible neighborhoods of L, not containing the empty clause. This proposition was proved in [AH15b] but we provide below a much simpler proof. Note that when a subset ψ' of an IS ψ is not necessary anymore [1], this property ensures that SUP can perform the same propagations in the transformed formula as in the original clauses of ψ'. More specifically, the UP-resilience of a set of literals L with respect to its neighborhood $neigh(L)$ is maintained individually on every literal in L.

Proposition 5.1. *Let ψ an IS and S be a sequence of variables appearing in ψ. For any set of literals L appearing in ψ, if the transformation $\Theta(\psi, S)$ is UP-resilient for L then $\forall N \in pneigh(L) : \square \in N$ or $\forall l \in L, l$ can be propagated in $\Theta(\psi, S)|_{N\setminus\{l\}}$.*

Proof. We prove this property by induction on $|L| = n$:

[1] for instance when ψ is not minimal

- If $n = 1$, then $L = \{l\}$ and the property is verified.
- Suppose the property is true for every set of size n. Let L be of size $n+1$ and l a literal in L. We set $L' = L \setminus \{l\}$ and let $N \in pneigh(L)$. Clearly, $N = N_1 \cup N_2$ where $N_1 \in pneigh(L')$ and $N_2 \in pneigh(l)$. Moreover, since $|L'| = n$, we know by induction that $\forall N \in pneigh(L') : \square \in N$ or every literal l' in L' can be propagated in $\Theta(\psi, S)|_{N\setminus\{l'\}}$. In particular, $\square \in N_1$ or every literal l' in L' can be propagated in $\Theta(\psi, S)|_{N_1\setminus\{l'\}}$. Also, The transformation $\Theta(\psi, S)$ is UP-resilient for L and particularly for l and thus, we have $\forall N \in pneigh(l) : \square \in N$ or l can be propagated in $\Theta(\psi, S)|_N$. In particular, $\square \in N_2$ or l can be propagated in $\Theta(\psi, S)|_{N_2}$. Thus, We have the following cases:
  - If $\square \in N_1$ or $\square \in N_2$ then $\square \in N$
  - Else, every literal l' in L' and l can be propagated respectively in $\Theta(\psi, S)|_{N_1\setminus\{l'\}}$ and $\Theta(\psi, S)|_{N_2}$. Therefore, the clauses that ensure the propagation of every literal l' in L' in $\Theta(\psi, S)|_{N_1\setminus\{l'\}}$ also ensure their propagation in $\Theta(\psi, S)|_{(N_1\cup N_2)\setminus\{l'\}}$ and, similarly, the clauses that ensure the propagation of l in $\Theta(\psi, S)|_{N_2}$ also ensure its propagation in $\Theta(\psi, S)|_{(N_1\cup N_2)\setminus\{l\}}$.

  We deduce that $\forall N \in pneigh(L) : \square \in N$ or every literal l in L can be propagated in $\Theta(\psi, S)|_{N\setminus\{l\}}$.

■

5.3 UP-Resilience and IS Transformation

An important factor that has a direct impact on the UP-resilience property is the Max-SAT resolution application order. Recall that two generic orders have been used in BnB solvers for Max-SAT, i.e., the Reverse Propagation Order (RPO) and the Smallest Intermediary Resolvent (SIR) order [AH14a] [2], and were empirically shown to influence the UP-resilience of transformations [AH15b]. In the following example, we exhibit this observation by showing that application orders can have a direct impact on the UP-resilience of certain literals appearing in the detected IS and therefore on the UP-resilience of the Max-SAT resolution transformation of an IS.

Example 5.1. *We consider the same IS* $\psi = \{x_1,\ x_2,\ x_3,\ \overline{x_3} \vee \overline{x_4},\ \overline{x_1} \vee \overline{x_2} \vee x_4\}$ *in Example 3.10, whose distinct implication graphs corresponding to the possible propagation sequences are represented in Figure 3.5. In Example 3.11, we showed that the transformation of* ψ *with respect to RPO is not UP-resilient since, in particular, it is not UP-resilient for literal* x_1*. Now, we consider the Max-SAT resolution transformation of* ψ *with respect to the variable sequence* $S = \langle x_3, x_4, x_1, x_2\rangle$ *corresponding to SIR which is given in Figure 5.1. We have* $\Theta(\psi, S) = \{x_1 \vee x_2,\ x_1 \vee x_4,\ x_3 \vee x_4,\ \overline{x_1} \vee x_2 \vee x_4,\ \square\}$*. The empty clause appears in the possible neighborhoods of the literals* x_4 *and* $\overline{x_4}$*. The propagation of the literal* x_3 *with respect to its possible neighborhood* $\{\overline{x_4}\}$*, not containing the empty clause, is ensured by the clause* $x_3 \vee x_4$*. Furthermore, the propagation of literals* x_1 *and* x_2 *with respect to their possible*

[2] refer to Section 3.2.1.5

neighborhood $\{x_4\}$*, not containing the empty clause, is ensured by the clauses* $x_1 \vee \overline{x_4}$ *and* $\overline{x_1} \vee x_2 \vee \overline{x_4}$*. Therefore, the transformation* $\Theta(\psi, S)$ *is UP-resilient.*

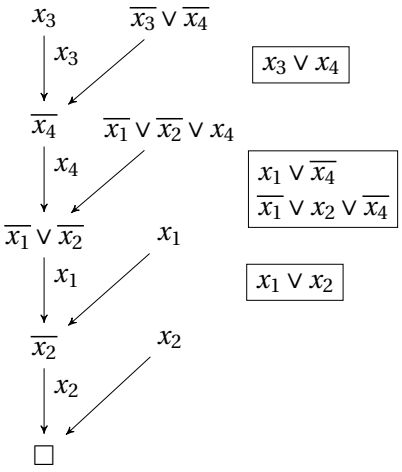


Figure 5.1: Implication graph corresponding to the Max-SAT resolution transformation of ψ with respect to SIR

5.4 UP-Resilience and Traditional Patterns

The UP-resilience property can help explain the efficiency of learning schemes in the literature. In particular, Proposition 3.2 establishes the UP-resilience of the traditional patterns P_1, P_2 and P_3 presented in Section 3.2.1.4 with respect to a specific ordering. However, no formal proof was provided for this proposition in [AH15b] whereas a proof for pattern P_1 only has been provided in [Abr15]. Hereafter, we provide formal proofs for the UP-resilience of patterns P_1, P_2 and P_3. In particular, unlike the result of Proposition 3.2, we show that any order of application consistent with these patterns ensures a UP-resilient transformation. This fact provides further insight on the relevance of the UP-resilience property in characterizing Max-SAT resolution transformations in Max-SAT BnB and in explaining the efficiency of the major patterns observed in the last decade. In the next section, we will also study the relation between UP-resilience and the recently introduced UCS patterns in [AH14d].

Proposition 5.2. *Let* ψ *be an IS and* $\psi' \subset \psi$ *bet a set matching the premises of pattern* P_1*. Then, the Max-SAT resolution transformation described in* P_1 *is UP-resilient.*

Proof. $\psi' = \{l_1 \vee l_2,\ l_1 \vee \overline{l_2}\}$. Therefore, there are two possible propagation sequences whose implication graphs are represented in Figure 5.2. Since all possible neighborhoods of literals $\overline{l_1}$, l_2 and $\overline{l_2}$ contain the empty clause, the transformation of ψ' as in P_1 with respect to the only possible variable sequence $S =< var(l_2) >$ is UP-resilient.

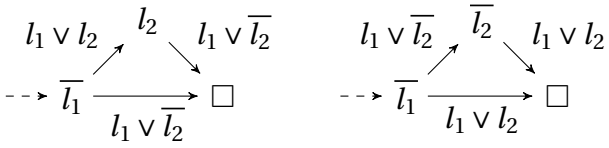


Figure 5.2: Implication graphs corresponding to the possible propagation sequences for an IS containing the premises of pattern P_1

■

Proposition 5.3. *Let ψ be an IS and $\psi' \subset \psi$ bet a set matching the premises of pattern P_2. Then, the Max-SAT resolution transformation described in P_2 is UP-resilient.*

Proof. $\psi' = \{l_1 \vee l_2,\ l_1 \vee l_3,\ \overline{l_2} \vee \overline{l_3}\}$. Therefore, there are two possible propagation sequences whose implication graphs are represented in Figure 5.3. There are two Max-SAT resolution application orders $S_1 = < var(l_2), var(l_3) >$ and $S_2 = < var(l_3), var(l_2) >$ that produce the same transformation described by pattern (P_2). Since all possible neighborhoods of l_2 and $\overline{l_2}$ contain the empty clause, the transformation of ψ by Max-SAT resolution is UP-resilient for l_2 and $\overline{l_2}$. We have $pneigh(\overline{l_1}) = \{\{l_3, \square\} \cup pred(\overline{l_1}), \{l_2, l_3\} \cup pred(\overline{l_1})\}$, where $pred(\overline{l_1}))$ denotes the predecessors of $\overline{l_1}$, and clearly the clause $C = \overline{l_1} \vee \overline{l_2} \vee \overline{l_3}$ propagates $\overline{l_1}$ when the literals l_2, l_3 in its second neighborhood are set to True. Also, $pneigh(l_3) = \{\{\overline{l_1}, \square\}, \{\overline{l_1}, \overline{l_2}\}\}$ and similarly the clause $C' = l_1 \vee l_2 \vee l_3$ propagates l_3 when the literals in its neighborhood $\{\overline{l_1}, \overline{l_2}\}$ are set to True.

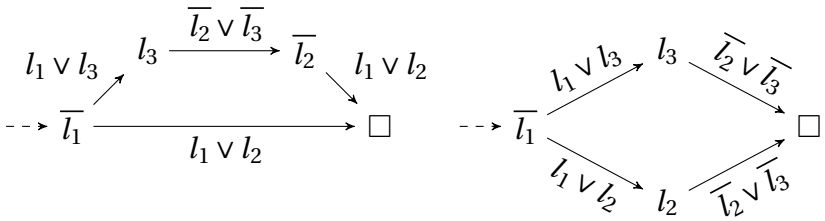


Figure 5.3: Implication graphs corresponding to the possible propagation sequences for an IS containing the premises of pattern P_2

■

Proposition 5.4. *Let ψ an IS that matches the premises of pattern P_3. Then, the Max-SAT resolution transformation described in P_3 is UP-resilient.*

Proof. $\psi = \{l_1,\ \overline{l_1} \vee l_2,\ \overline{l_2} \vee l_3, ..., \overline{l_{k-1}} \vee l_k,\ \overline{l_k}\}$. Therefore, there are k possible propagation sequences whose implication graphs are represented in Figure 5.4. In each graph a different clause of ψ, containing the literal l_i where $1 < i < k$ is falsified. When $i = 1$ (resp. $i = k$), the unit clause l_i (resp. $\overline{l_k}$) is falsified. We have $pneigh(l_i) = \{\{l_{i-1}, l_{i+1}\}, \{l_{i-1}, \square\}\}$ for $1 < i < k$ and, clearly, the clause $l_i \vee \overline{l_{i+1}}$, obtained after the transformation, ensures the propagation

of the literal l_i with respect to its neighborhood $\{l_{i-1}, l_{i+1}\}$. Also, for literals l_1 and l_k, we have respectively $pneigh(l_1) = \{\{l_2\}, \{\square\}\}$ and $pneigh(l_{k+1}) = \{\{l_k, \square\}\}$ and, clearly, the clause $l_1 \vee \overline{l_2}$ ensures the propagation of the literal l_1 with respect to its neighborhood $\{l_2\}$. The same arguments ensure UP-resilience for literals $\overline{l_i}$ for $1 \leq i \leq k$. It is important to note that this proof on the UP-resilience of ISs matching the premises of pattern P_3 is clearly valid for every Max-SAT resolution application order consistent with P_3.

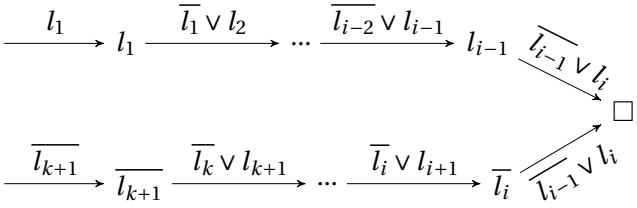


Figure 5.4: Implication graphs corresponding to the possible propagation sequences for an IS matching the premise of pattern P_3

■

5.5 UP-Resilience and UCS Patterns

In this section, we study the relation between UP-resilience and the recently introduced UCS patterns in [AH14d]. We prove that the Max-SAT resolution transformation of binary UCSs is UP-resilient with respect to two newly introduced orders. This result is established through a characterization of the implication graphs corresponding to propagation sequences leading to the detection of such patterns. We also show that, unlike the devised orders, the current used mechanisms can't ensure UP-resilience for binary UCSs thus providing an explanation to the recent empirical results established in [AH14d] and showing that UP-resilience can help extend the current patterns used in state-of-the-art solvers. Furthermore, we generalize our result on the resilience of k^b-UCSs to UCSs where all clauses are binary except one of any size that is involved in the conflict.

5.5.1 On Implication Graphs of UCS Patterns

In this section, we establish a characterization of implication graphs of the studied k-UCSs. More specifically, we study the portion of the graph representing the propagation sequence of the detected k-UCS which is delimited by the FIUP and the empty clause. This characterization is established in the following lemma for k-UCS whose clauses are binary except for the conflict clause which can be of any size $s \geq 2$.

Lemma 5.1. *Let $k \geq 2$ and ψ be a k-UCS whose clauses are binary except for the conflict clause of size $s \geq 2$, recognized by the FUIP l in an implication graph G of an IS such that $|succ(l)| = s$. Then, there exists exactly s disjoint paths from l to $\square$ in G.*

Proof. Since l is a UIP, all the paths from the literals propagated by unit clauses to the conflict node in G pass through it. We have $|succ(l)| = s$. Therefore, there are at least s different paths from l to $\square$ in G. Let $p_1,...,p_s$ be those paths. Suppose we have a different path p_{s+1} from l to $\square$. We have two possible cases:

- $|pred(\square)| \neq s$. This is absurd since the conflict clause is of size s and thus $|pred(\square)| = s$.
- Else, since $|pred(\square)| = s$, there exists $l' \neq l \in p_{s+1}$ and $i \in \{1,...,s\}$ such that $l' \in p_i$ and $|pred(l')| > 1$. This is absurd since all clauses of ψ except the conflict clause are binary.

We deduce that there are exactly s different paths from l to $\square$ in G. The same argument of the second case ensures that these paths are disjoint.

■

As explained in Section 3.2.1.5, when a UCS is detected, we know that the reverse propagation order ensures the production of a unit resolvent clause after the transformation. However, in general, this is not necessarily true for all application orders. Since this is the main feature of UCS patterns, we must ensure that the introduced orders produce a unit resolvent clause. It is important to note that the condition on the successors of the FUIP in Lemma 5.1 ensures the production of such clause for all possible orders. In the next section, we prove the UP-resilience of binary k-UCSs. To this end, we show in the next proposition that the condition on the FUIP successors in Lemma 5.1 is always verified for binary k-UCSs. Later, when we generalize our result on further UCS patterns, we only consider the graphs described by Lemma 5.1, i.e., which verify the condition on the successors of the FUIP.

Proposition 5.5. *Let $k \geq 2$ and ψ be a k^b-UCS recognized by the FUIP l in an implication graph G of an IS. Then, $|succ(l)| = 2$.*

Proof. Suppose that $|succ(l)| \neq 2$. We have two possible cases:

- if $|succ(l)| > 2$ then, since $|succ(\square)| = 2$, there exists a literal with two predecessors. This is absurd since all the clauses are binary.
- if $|succ(l)| = 1$ then l is not the FUIP which is absurd.

■

Corollary 5.1. *Let $k \geq 2$ and ψ be a k^b-UCS recognized by the FUIP l in an implication graph G of an IS. There exists exactly two disjoint paths from l to $\square$ in G.*

Proof. Result trivially implied by Lemma 5.1 and Proposition 5.5.

■

5.5.2 On the UP-Resilience of Binary UCSs

In Section 5.5, we proved the UP-resilience of the main patterns P_1 and P_2 with respect to any order of application of Max-SAT resolution. The next corollary is an immediate consequence of this result.

Corollary 5.2. *For $k \in \{2,3\}$, the transformation of a k^b-UCSs is UP-resilient.*

Proof. 2^b-UCSs and 3^b-UCSs are all of the respective forms $\psi_{2^b} = \{l_1 \vee l_2,\ l_1 \vee \overline{l_2}\}$ and $\psi_{3^b} = \{l_1 \vee l_2,\ l_1 \vee l_3,\ \overline{l_2} \vee \overline{l_3}\}$ which correspond to the premises of patterns P_1 and P_2. Thus, we obtain the wanted result using Propositions 5.2 and 5.3. ■

We want to generalize the result of Corollary 5.2 to all binary k-UCSs, even for specific orders of application of Max-SAT resolution. To this end, we introduce new orders using the characterization established in Corollary 5.1. We prove the resilience of k^b-UCSs with respect to these orders. In the following section, we will also explain the inefficiency of the current orders for these patterns.

Definition 5.1 (Path Resolvent Order)**.** *Let $p_1 = \langle l, l_1^{p_1}, ..., l_{n_1}^{p_1}, \square\rangle$ and $p_2 = \langle l, l_1^{p_2}, ..., l_{n_2}^{p_2}, \square\rangle$ where $n_1, n_2 \geq 1$ denote two disjoint paths from l to $\square$. The Path Resolvent Order (PRO) of p_1 and p_2 is defined as follows:*

$$PRO(p_1, p_2) = \langle var(l_1^{p_1}), ..., var(l_{n_1}^{p_1}), var(l_1^{p_2}), ..., var(l_{n_2}^{p_2})\rangle$$

Theorem 5.1. *For any $k \geq 2$, the transformation of a k^b-UCSs with respect to PRO is UP-resilient.*

Proof. Let $k \geq 2$ and ψ be a k^b-UCS recognized by the FUIP l in the implication graph G of an IS. By Corollary 5.1, we know that there are 2 disjoint paths from l to $\square$ in G. Let $p_1 = \langle l, l_1^{p_1}, ..., l_{n_1}^{p_1}, \square\langle (n_1 \geq 0)$ and $p_2 = \langle l, l_1^{p_2}, ..., l_{n_2}^{p_2}, \square\rangle (n_2 \geq 0)$ denote these paths in G where $n_1 + n_2 = k - 1$. And, suppose w.l.o.g that $l_{n_1}^{p_1} = l'$ is the conflict literal, i.e., the last propagated literal. We have two possible propagation sequences whose implication graphs are G and G' represented in Figure 5.5.

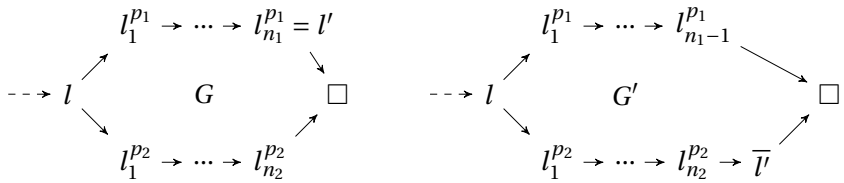


Figure 5.5: Implication graphs corresponding to the possible propagation sequences for k^b-UCSs.

We prove that the Max-SAT resolution transformation relatively to the order $O = PRO(p_1, p_2)$ is UP-resilient:

- The clause propagating l is not deleted after the transformation by Max-SAT resolution relatively to the order O so it clearly propagates l if its predecessors are set to True and thus the transformation by Max-SAT resolution relatively to the order O is UP-resilient for l. This argument also applies for the literals that were involved in the propagation of l.
- All possible neighborhoods of literals $l_{n_1}^{p_1} = l'$ and $\overline{l'}$ contain the empty clause. Therefore, the transformation by Max-SAT resolution relatively to the order O is UP-resilient for l' and $\overline{l'}$.
- For $i \in \{1,2\}$, we set $l_0^{p_i} = l$. Every literal $l_j^{p_i}$ such that $1 \leq j < n_i$ admits exactly one neighborhood $neigh(l_j^{p_1}) = \{l_{j-1}^{p_i}, l_{j+1}^{p_i}\}$ that doesn't contain the empty clause. Similarly, for $l_{n_2}^{p_2}$, we have $neigh(l_{n_2}^{p_2}) = \{l_{n_2-1}^{p_i}, \overline{l'}\}$. The Max-SAT resolution step on $var(l_j^{p_1})(1 \leq j < n_i)$ is of the form:

$$\frac{\overline{l} \vee l_j^{p_i},\ \overline{l_j^{p_i}} \vee l_{j+1}^{p_i}}{\overline{l} \vee l_{j+1}^{p_i},\ \overline{l} \vee l_j^{p_i} \vee \overline{l_{j+1}^{p_i}},\ l \vee \overline{l_j^{p_i}} \vee l_{j+1}^{p_i}}$$

  The clause $C = l \vee \overline{l_j^{p_i}} \vee l_{j+1}^{p_i}$ clearly ensures the propagation of literal $l_{j+1}^{p_i}$ if $l_j^{p_i} \in neigh(l_{j+1}^{p_i})$ is set to True since $\overline{l}$ is propagated by the unit resolvent clause $\{\overline{l}\}$. Also, for $j = 1$, the clause $C' = \overline{l} \vee l_1^{p_1} \vee \overline{l_2^{p_1}}$ ensures the propagation of $l_1^{p_1}$ if $l, l_2^{p_1} \in neigh(l_1^{p_1})$ are set to True. Thus, We deduce that the transformation is UP-resilient for $l_j^{p_i}$ where $1 \leq j \leq n_i$ $(j \neq n_1)$.

We conclude that the transformation of ψ by Max-SAT resolution relatively to the order O is UP-resilient.

■

Definition 5.2 (Path Resolvent Circular Order)**.** *Let $p_1 = \langle l, l_1^{p_1}, ..., l_{n_1}^{p_1}, \square\rangle$ and $p_2 = \langle l, l_1^{p_2}, ..., l_{n_2}^{p_2}, \square\rangle$ where $n_1, n_2 \geq 1$) denote two disjoint paths from l to $\square$. The Path Resolvent Circular Order (PRCO) of p_1 and p_2 is defined as follows:*

$$PRCO(p_1, p_2) = \langle var(l_1^{p_1}), ..., var(l_{n_1}^{p_1}), var(l_{n_2}^{p_2}), ..., var(l_1^{p_2})\rangle$$

Theorem 5.2. *For any $k \geq 2$, the transformation of k^b-UCSs with respect to PRCO is UP-resilient.*

Proof. Let $k \geq 2$ and ψ be a k^b-UCS recognized by the FUIP l in the implication graph G of an IS. By corollary 5.1, let $p_1 = \langle l, l_1^{p_1}, ..., l_{n_1}^{p_1}, \square\rangle (n_1 \geq 0)$ and $p_2 = \langle l, l_1^{p_2}, ..., l_{n_2}^{p_2}, \square\rangle (n_2 \geq 0)$ denote the two disjoint paths from l to $\square$ in G where $n_1 + n_2 = k - 1$. And, suppose w.l.o.g that $l_{n_1}^{p_1} = l'$ is the conflict literal. We have two possible propagation sequences whose implication graphs are G and G' represented in Figure 5.5. We prove that the Max-SAT resolution transformation relatively to the order $O = PRCO(p_1, p_2)$ is UP-resilient:

- The same arguments in the proof of Theorem 5.1 ensure the UP-resilience of the transformation respectively to O for $l_j^{p_1}$ $(1 \leq j \leq n_1)$ and $\overline{l'}$ as well as l and all the literals involved in its propagation.
- Every literal $l_j^{p_2}$ such that $1 \leq j \leq n_2$ admits exactly one neighborhood $neigh(l_j^{p_2}) = \{l_{j-1}^{p_2}, l_{j+1}^{p_2}\}$ that doesn't contain the empty clause (we set $l_0^{p_2} = l$ and $l_{n_2+1}^{p_2} = \overline{l'}$). The Max-SAT resolution step on $var(l_j^{p_2})$ $(j \neq 1)$ is of the form:

$$\frac{\overline{l} \vee \overline{l_j^{p_2}},\ l_j^{p_2} \vee \overline{l_{j-1}^{p_2}}}{\overline{l} \vee \overline{l_{j-1}^{p_2}},\ \overline{l} \vee \overline{l_j^{p_2}} \vee l_{j-1}^{p_2},\ l \vee l_j^{p_2} \vee \overline{l_{j-1}^{p_2}}}$$

  The clause $C = l \vee l_j^{p_2} \vee \overline{l_{j-1}^{p_2}}$ clearly ensures the propagation of literal $l_j^{p_2}$ when $l_{j-1}^{p_2} \in neigh(l_j^{p_2})$ is set to True since $\overline{l}$ is propagated by the unit resolvent clause $\{\overline{l}\}$. Also, the clause $C' = \overline{l} \vee \overline{l_2^{p_2}} \vee l_1^{p_2}$, generated by the Max-SAT resolution step on $var(l_2^{p_2})$, clearly ensures the propagation of $l_1^{p_2}$ when its neighbors $l, l_2^{p_2} \in neigh(l_1^{p_2})$ are set to True. Thus, the transformation is UP-resilient for $l_j^{p_2}$ where $1 \leq j \leq n_2$.

We conclude that the transformation by Max-SAT resolution relatively to the order O is UP-resilient.

■

There is a major difference between the orders we introduced. Indeed, PRCO ensures a linear input resolution transformation, i.e., at each intermediary Max-SAT resolution step we use the resolvent obtained in the previous step and a clause from the detected k^b-UCS. This is not always the case for PRO. The following result is an immediate consequence of either Theorem 5.1 or 5.2.

Corollary 5.3. *For any $k \geq 2$, there exists a sequence of variables S in any k^b-UCSs ψ such that $\Theta(\psi, S)$ is UP-resilient.*

5.5.3 On the Limits of Current Orders for Binary UCSs

Empirical results show that 2^b-UCSs and 3^b-UCSs, which correspond respectively to the patterns P_1 and P_2 have a positive impact on the performance of BnB solvers for Max-SAT [AH14d; Li+10a]. The result in corollary 5.2 obtained through properties 5.2 and 5.3 prove that 2^b-UCSs and 3^b-UCSs are UP-resilient for any given order of application of Max-SAT resolution which explains why learning them has a positive impact regardless of the chosen order. This is not the case for k^b-UCSs when $k > 3$. Empirical studies on the Ahmaxsat solver in [AH14d] show that learning 4^b-UCSs and 5^b-UCSs can have a negative impact on its performance. This can be explained by the inadequacy of the Max-SAT resolution application orders used in state-of-the-art BnB solvers for k^b-UCSs when $k > 3$. This result is established in the following propositions.

Proposition 5.6. *For any $k > 3$, there exists a k^b-UCS whose transformation with respect to RPO is not UP-resilient.*

Proof. We provide a counter example for 4^b-UCSs which can be easily extended to any k^b-UCS for $k > 4$. We consider the IS $\psi = \{l, \overline{l} \vee l_1, \overline{l} \vee l_2, \overline{l_1} \vee l_3, \overline{l_2} \vee \overline{l_3}\}$ detected by one of the possible implication graphs represented on the left in Figure 5.6 after the respective propagation of literals l_1, l_2 and l_3 (or $\overline{l_3}$). Clearly, the subset $\psi' = \{\overline{l} \vee l_1, \overline{l} \vee l_2, \overline{l_1} \vee l_3, \overline{l_2} \vee \overline{l_3}\} \subset \psi$ is a 4^b-UCS recognized by the FUIP l. The Max-SAT resolution transformation of ψ' with respect to RPO which corresponds to the variable sequence $S = \langle var(l_3), var(l_2), var(l_1) \rangle$ is represented on the right in Figure 5.6.

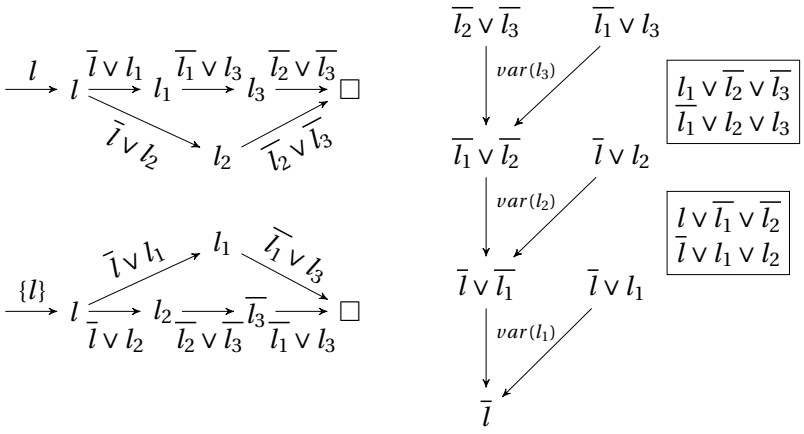


Figure 5.6: Implication graphs corresponding to the possible propagation sequences of ψ and the application of Max-SAT resolution steps relatively to RPO

The literal l_1 has one neighborhood $neigh(l_1) = \{l, l_3\}$ that doesn't contain the empty clause. Clearly, the literal l_1 can't be propagated in $\Theta(\psi, S)|_{neigh(l_1)} = \{l_1 \vee \overline{l_2}, l_1 \vee l_2\}$. Similarly, the fragmentation phenomenon also occurs for l_2 and we conclude that the transformation of ψ' relatively to RPO is not UP-resilient.

■

Proposition 5.7. *For any $k > 3$, there exists a k^b-UCSs whose transformation with respect to SIR is not UP-resilient.*

Proof. The SIR heuristic becomes practically unusable for k^b-UCSs since all the intermediary resolvents have the same size (binary) as shown in the proofs of Theorems 5.1 and 5.2. The same argument in the proof of Proposition 5.6 is therefore sufficient to prove the same result on the SIR order.

■

5.5.4 Generalization to Other UCSs

In this section, we generalize our result to k-UCSs where all clauses are binary except one of any size that is involved in the conflict when the implication graph corresponds to the

characterization established in Lemma 5.1. A clause involved in the conflict is either the falsified clause or contains the conflict literal, i.e., the last propagated literal. Unfortunately, although PRCO has the advantage of ensuring a linear input transformation, we couldn't generalize it to obtain the wanted result. Nevertheless, we managed to prove our result using a generalization of PRO to a multitude of paths.

Definition 5.3 (Multiple Path Resolvent Order)**.** *Let $s \geq 2$ and $p_1 = \langle l, l_1^{p_1}, ..., l_{n_1}^{p_1}, \square \rangle, ..., p_s = \langle l, l_1^{p_2}, ..., l_{n_s}^{p_s}, \square \rangle$ denote s disjoint paths from l to $\square$. The Multiple Path Resolvent Order (MPRO) of $p_1, ..., p_s$ is defined inductively on s as follows:*

- *If $s = 2$, $MPRO(p_1, p_2) = PRO(p_1, p_2)$*
- *Else $MPRO(p_1, ..., p_s) = PRO(\langle l, MPRO(p_1, ..., p_{s-1}), \square \rangle, p_s)$.*

Theorem 5.3. *Let $k \geq 2$ and ψ be a k-UCS whose clauses are binary except for the conflict clause C of size $|C| = s \geq 3$, recognized by the FUIP l in the implication graph G of an IS such that $|succ(l)| = s$. The transformation of ψ with respect to MPRO is UP-resilient.*

Proof. We suppose w.l.o.g that $C = \{\overline{l_1}, ..., \overline{l_s}\}$. By Lemma 5.1, there are exactly s disjoint paths $p_1 = \langle l, l_1^{p_1}, ..., l_{n_1}^{p_1}, \square \rangle, ..., p_s = \langle l, l_1^{p_s}, ..., l_{n_s}^{p_s}, \square \rangle$ from l to $\square$ in the implication graph G, represented in Figure 5.7, such that $\sum_{i=1}^{s} n_i = k - 1$ and $l_{n_i}^{p_i} = l_i$ for $i \in \{1, ..., s\}$. Other than G, there are exactly $\binom{s-1}{s} = s$ possible implication graphs all similar to the graph G' represented in Figure 5.7.

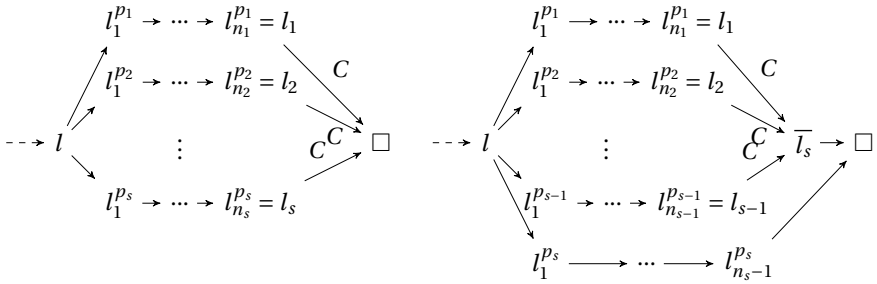


Figure 5.7: Implication graphs corresponding to the possible propagation sequences for k-UCSs with binary clauses except for the conflict clause

We prove that the Max-SAT resolution transformation relatively to the order $O = MPRO(p_1, ..., p_s)$ is UP-resilient:

- The same arguments in the proof of Theorem 5.1 ensure the UP-resilience of the transformation respectively to O for $l_j^{p_i}$ where $1 \leq i \leq s$ and $1 \leq j < n_i$ as well as l and all the literals involved in its propagation. Furthermore, all the neighborhoods of literals $\overline{l_1}, ..., \overline{l_s}$ contain the empty clause.
- For $i \in \{1, ..., s\}$, $\forall N \in pneigh(l_{n_i}^{p_i})$ $(n_i > 1)$ s.t. $\square \notin N$, we have $l_{n_i-1}^{p_i} \in N$ (exists since $n_i > 1$). Clearly, the clause $l \vee \overline{l_{n_i-1}^{p_i}} \vee l_{n_i}^{p_i}$ obtained by the application of Max-SAT

resolution on $var(l^{p_i}_{n_i-1})$ ensures the propagation of $l^{p_i}_{n_i}$ in any of these neighborhoods when $l^{p_i}_{n_i-1}$ is set to True since $\overline{l}$ is propagated by the unit resolvent clause containing it. We deduce that the transformation relatively to the order O is UP-resilient for $l^{p_i}_{n_i}$ where $1 \leq i \leq s$ and $n_i > 1$.

- We still need to prove the UP-resilience of the transformation for literals $l^{p_i}_{n_i} = l_i$ when $n_i = 1$, with respect to their possible neighborhoods $\{l, \overline{l_j}\}$ for $j \in \{1,..,s\} \setminus \{i\}$ not containing the empty clause. For this end, we prove by induction on $|C| \geq 3$ that the compensation clauses produced by the Max-SAT resolution steps on $var(l_1),...,var(l_s)$ ensure the propagation of each literal l_i if we consider the neighborhoods as mentioned above. For simplification, in the first Max-SAT resolution step, we replace C by the clause $C' = \overline{l} \vee \overline{l_1} \vee \cdots \vee \overline{l_s}$. This doesn't affect our result since we only omit a single clause containing the literal l:
  - If $|C| = 3$ then $C = \overline{l_1} \vee \overline{l_2} \vee \overline{l_3}$. The Max-SAT resolution steps are represented on the left in Figure 5.8 and we can easily check that the compensation clauses ensure the propagation of the literals l_i, for $1 \leq i \leq 3$, if we consider the neighborhoods mentioned above.
  - Suppose the property is true for any clause of size $s \geq 3$. Let $C = l_1 \vee \cdots \vee l_{s+1}$ be a clause of size $s+1$. The first Max-SAT resolution step is represented on the right in Figure 5.8. The resolvent clause is $\overline{l} \vee l_2 \vee \cdots \vee l_{s+1}$ and if we consider $C' = l_2 \vee \cdots \vee l_{s+1}$ of size s we ensure by induction the propagation of any literal l_i where $2 \leq i \leq s+1$ with respect to the neighborhoods $\{l, \overline{l_j}\}$ for $j \in \{2,..,s+1\} \setminus \{i\}$. Thus, each compensation clause $CC_k = \overline{l} \vee l_1 \vee \overline{l_2} \vee \cdots \vee \overline{l_k} \vee l_{k+1}\}$ for $k \in \{1,...,s\}$ ensures the propagation of literal l_1 with respect to the neighborhood $\{l, \overline{l_{k+1}}\}$ since, by induction, the propagation of literals $l_2,...,l_k$ is ensured in the same neighborhood. Now, we prove by induction on $k \in \{1,...,s\}$ that the clause CC_k ensures the propagation of l_{k+1} with respect to the neighborhood $\{l, \overline{l_1}\}$:
    - If $k = 1$, $CC_1 = \overline{l} \vee l_1 \vee l_2$ clearly ensures the propagation of l_2 with respect to the neighborhood $\{l, \overline{l_1}\}$.
    - Suppose for $1 \leq k' < k \leq s$, $CC_{k'}$ ensures the propagation of $l_{k'+1}$ with respect to the neighborhood $\{l, \overline{l_1}\}$. $CC_k = \overline{l} \vee l_1 \vee \overline{l_2} \vee \cdots \vee \overline{l_k} \vee l_{k+1}$ clearly ensures the propagation of literal l_{k+1} with respect to the neighborhood $\{l, \overline{l_1}\}$ since, by induction, the propagation of $l_2,...,l_k$ is ensured in the same neighborhood by the clauses $CC_1,...,CC_{k-1}$.

We conclude that the transformation by Max-SAT resolution relatively to the order O is UP-resilient.

■

Corollary 5.4. *Let $k \geq 2$ and ψ be a k-UCS whose clauses are binary except for a single clause C of size $|C| = s \geq 3$ involved in the conflict, recognized by the FUIP l in the implication graph G of an IS such that $succ(l) = s$. There exists a UP-resilient transformation of ψ.*

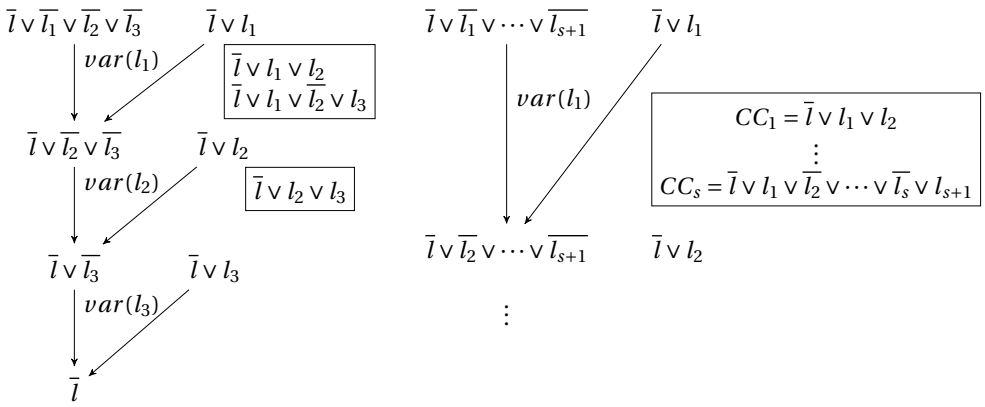


Figure 5.8: Application of Max-SAT resolution steps on the variables of the non binary clause C by induction on its size

Proof. If C is the conflict clause then we obtain the result by Theorem 5.3. Else, C contains the conflict literal and the detected implication graph G has the same form as the second graph represented in Figure 5.7. Clearly, there is a propagation sequence where C is falsified, i.e., corresponding to an implication graph G' similar to the first graph represented in Figure 5.7. Thus, we deduce the UP-resilience of the transformation with respect to MPRO through the same arguments in the proof of Theorem 5.3. ■

The SIR order is defined relatively to the size of the intermediary resolvents. Thus, it may theoretically simulate any order when the sizes of the resolvents are the same or many different orders when many resolvents share the same size which is the case of the studied UCSs. That's why this heuristic remains practically unusable even in the generalized case. Furthermore, RPO doesn't necessarily ensure the UP-resilience of k-UCSs described in the previous corollary. We finish this section by an example that highlights this fact. This example where the non binary clause is tertiary can be easily extended to any size $s > 3$.

Example 5.2. *We consider the IS* $\psi = \{l, \overline{l} \vee l_1, \overline{l} \vee l_2, \overline{l} \vee l_3, \overline{l_1} \vee l_4, \overline{l_2} \vee \overline{l_3} \vee \overline{l_4}\}$ *(we name the tertiary clause* C*) detected by the first implication graph represented on the left in Figure 5.9 after the respective propagation of literals* l_1, l_2, l_3 *and* l_4*. In the second graph on the left in the same figure, we represent another possible propagation sequence which outlines the possible neighborhood of* l_4*,* $neigh(l_4) = \{l_1, \overline{l_3}\}$ *not containing the empty clause. Clearly, the subset* $\psi' = \psi \setminus \{l\}$ *is a* 5*-UCS recognized by the FUIP* l *such that* C *participates in the conflict and* $|succ(l)| = |C| = 3$*. The Max-SAT resolution transformation of* ψ' *with respect to RPO which corresponds to the variable sequence* $S = \langle var(l_4), var(l_3), var(l_2), var(l_1)\rangle$ *is represented on the right in Figure 5.9. Clearly, the literal* l_4 *can't be propagated in* $\Theta(\psi, S)|_{neigh(l_4)} = \{\overline{l}, \overline{l} \vee \overline{l_2}, l_2 \vee l_4, \overline{l_2} \vee l_4\}$*. We conclude that the transformation of* ψ' *relatively to RPO is not UP-resilient.*

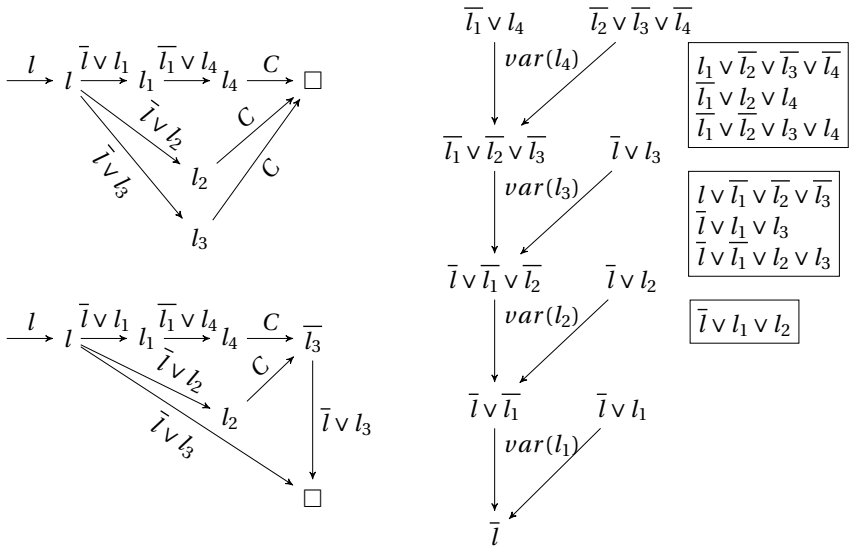


Figure 5.9: Implication graphs corresponding to the possible propagation sequences of ψ in Example 5.2 and the application of Max-SAT resolution steps relatively to RPO

5.6 Conclusion & Future Work

In this chapter, we have thoroughly investigated the power of inference in the context of Max-SAT BnB through the property of UP-resilience. We showed that this notion can help quantify the impact of Max-SAT resolution transformations on the SUP mechanism. We particularly observed that the order of application of Max-SAT resolution during the transformation of ISs can directly impact its UP-resilience. We showed that, while the traditional patterns are UP-resilient with respect to any order, the recently introduced k-UCS patterns are not necessarily UP-resilient with respect to the current orders specifically when $k > 3$. They are nevertheless UP resilient with respect to other newly introduced orders. Our results thus contribute to theoretically explain the efficiency of the traditional patterns observed in the literature as well as the recent empirical results established in [AH14d; Abr15].

To our best knowledge, this is the first work in which UP-resilience is used to theoretically characterize the transformations by Max-SAT resolution with respect to the possible orders of its application. Indeed, this can be a starting point of a new approach to extend Max-SAT resolution patterns. In our case, we chose UCS patterns because they present several advantages, i.e., the introduction of unit clauses as well as the high frequency of their apparition. More importantly, the specific UCS patterns studied in this chapter exhibit a particular structure in terms of the implication graphs representing their possible propagation sequences, which we exploited to introduce our new orders ensuring the UP-resilience of Max-SAT resolution transformations.

The prospects of our research include the extension of our studies to k-UCSs in general. It also opens new perspectives for finding orders of application of Max-SAT resolution that

ensure UP-resilience or maximize its percentage by thoroughly studying the implication graphs corresponding to the propagation sequences of certain ISs. Another interesting prospect is extending our work on more sophisticated mechanisms that are used to detect disjoint ISs such as Generalized Unit Propagation (GUP) [Kue12] which combines SUP whith Failed Literals (FL) [LMP06]. For instance, if the reason of unsatisfiablity of the detected IS after applying GUP can be represented in the form of an implication graph, our results would be applicable for GUP. Moreover, it would be relevant to study whether Max-SAT inference under the Max-SAT resolution rule can be efficiently incorporated in the recent BnB paradigm introduced in [Li+21a; Li+22a] which seems to bring forth a new family of BnB algorithms that rely on extensive learning. The extended local learning mechanisms deduced in our work may therefore allow to further guide the exploration of the search tree if properly incorporated during the search or can simply be used in the context of preprocessing. Finally, increasing knowledge about Max-SAT resolution can be useful for SAT-based solvers as some solvers [3] such as EVA [NB14], already exploit Max-SAT resolution to transform cores returned by SAT solvers.

[3] refer to Section 3.2.2

6 Bridging the Gap between SAT and Max-SAT Inference

Contents

6.1 Introduction & Motivation

One of the first proof systems for Max-SAT is based on the Max-SAT resolution inference rule [BLM06; LH05a; BLM07; LHG08], which is an extension of the resolution rule introduced in the context of SAT [Rob65]. Max-SAT resolution is sound and complete for Max-SAT and has recently been extensively studied in the context of proof theory for Max-SAT [LR20b; LR20a; BL20; Fil+20]. However, we still lack understanding of many aspects related to Max-SAT resolution as an inference rule and as a proof system. Our contributions in Chapter 5 provides a more through understanding on the power of Max-SAT resolution as an inference rule in the context Max-SAT BnB algorithms. In this chapter, we tackle more generic open problems and investigate some aspects related to Max-SAT resolution, potentially augmented with other relevant rules, as a proof system for Max-SAT.

SAT and Max-SAT are strongly related and share many aspects. In fact, SAT solving techniques are often used in the context of Max-SAT solving, particularly in SAT-based and

Branch and Bound (BnB) algorithms for Max-SAT [1]. Yet, in theory, bridging the gap between SAT and Max-SAT inference remains one of the main challenges in the last decade. In particular, adapting a resolution proof to get a valid proof for Max-SAT of reasonable size remains an open problem since the introduction of Max-SAT resolution 15 year ago. Bonet et al. state that "*it seems difficult to adapt a classical resolution proof to get a Max-SAT resolution proof, and it is an open question if this is possible without increasing substantially the size of the proof*" [BLM07]. Indeed, unlike resolution, the Max-SAT resolution rule replaces the premises with the conclusions, which is necessary to maintain Max-SAT equivalence after its application. Moreover, aside from the traditional resolvent clause, additional compensation clauses are also added to ensure Max-SAT equivalence. In [LHG08], Larrosa et al. describe Max-SAT resolution as "*a movement of knowledge*" and this is typically expected of any sound Max-SAT inference rule preserving Max-SAT equivalence. Many complications thus arise when adapting resolution proofs into Max-SAT resolution proofs. More specifically, read-once resolution proofs, where each clause is used once, represent the only fragment of resolution for which an immediate and trivial adaptation is possible [2]. Our first contributions in this chapter fall within this context and particularly aim to bridge the gap between SAT and Max-SAT inference. We propose different adaptations from resolution refutations into max-refutation where Max-SAT resolution is augmented with the split rule. We show that, in this case, we can obtain linear adaptations with respect to the size of the resolution refutations in the following cases: regular tree resolution, tree resolution and semi-tree resolution. We also generalize our results to unrestricted resolution refutations but with a worst case exponential blow up in the size of the proofs. Then, we tackle the more specific and difficult problem of adapting resolution refutations into Max-SAT resolution proofs without any additional inference rules. More specifically, we identify a new fragment of resolution, called crossing-free resolution, for which an adaptation using exclusively Max-SAT resolution is possible without substantially increasing the size of the proof. Our adaptations led to the the introduction of the first independent proof builder for Max-SAT, called MS-Builder, which relies on iterative calls to a SAT oracle in order to generate certificates. Our work also tackles a more generic aspect on proof systems for Max-SAT. We particularly introduce and investigate the notion of explainability to study the inferential power of Max-SAT proof systems.

This chapter is organized as follows. First, we propose our different adaptations aiming to bridge the gap between SAT and Max-SAT inference. We first describe our adaptations to max-refutations, where Max-SAT resolution is augmented with the split rule, ensuring linear size guarantees for specific classes of resolution. Then, we introduce our new class of resolution for which an adaptation using exclusively Max-SAT resolution is possible without a considerable increase in the size of the proofs. We also study the limits and features of our adaptations with respect to specific resolution patterns. We also briefly describe further results on certificate generation and on explainability for Max-SAT. Finally, we mention that our contributions in this chapter are in collaboration with Matthieu Py. These results have been the subject of joint publications in different international [PCH20; PCH21a; PCH21d;

[1] refer to Section 3.2
[2] Refer to Section 3.3.2

CHP22b] and national [PCH21c; PCH22a; PCH22b; CHP22a] venues. The results succinctly described in Sections 6.2 and 6.5 are presented with further details in the manuscript of Mathieu Py [Py21].

6.2 From SAT Refutations to Max-Refutations

In this section, we attempt to tackle the problem of adapting resolution refutations into max-refutations with size guarantees on the resulting proofs. Note that, in these results, we augment Max-SAT resolution with the split rule thus generating Max-SAT proofs in the ResS system [LR20b]. We propose linear adaptations for specific classes of resolution. We first deal with regular tree resolution refutations, showing that a linear adaptation is possible in this case by applying the split rule on to duplicate non-read-once clauses when necessary. Then, we extend this result to tree resolution refutations using the known result established in Lemma 2.1 stipulating that minimal tree resolution refutations are regular. Furthermore, we introduce a new class of refutations that we refer to as semi-tree-like, which generalizes tree resolution refutations, and we extend our linear result to this class of refutations. Finally, we propose a complete adaptation of any (unrestricted) resolution refutation into a max-refutation, although with a worst case exponential blow-up in the size of the resulting refutations.

6.2.1 From Regular Tree Resolution Refutations to Max-Refutations

In this section, we show that it is possible to adapt a regular tree resolution refutation to obtain a max-refutation with linear size in ResS. If a clause C is used k times ($k > 1$) as a premise of a resolution step, we use the split rule to duplicate clause C into k distinct clauses subsumed by C. We will then use these new clauses to replace C as a premise of a resolution step. Given a branch starting from a clause C, we say that this branch accepts the substitution of C by $C \vee l$ if updating the branch after the substitution of C by $C \vee l$ does not affect the validity of the resolution refutation. The following lemma guarantees that, for a given non-read-once clause, there exists a variable x such that some branches starting from C accept the substitution of C by $C \vee x$ while the rest accept the substitution of C by $C \vee \overline{x}$. Note that a node where a set of given paths in a resolution proof intersect will be referred to as their junction node.

Lemma 6.1. *Given a non-read-once regular tree resolution refutation π and a non-read-once clause C in π, there exists a variable $x \notin var(C)$ such that it is possible to partition the branches starting from C into two non-empty subsets of branches, the branches in the first subset accepting the substitution of C by $C \vee x$ and the branches in the second accepting the substitution of C by $C \vee \overline{x}$.*

Proof. Given a non-read-once regular tree resolution refutation π and a non-read-once clause C in π, there exists a node v of the DAG of π representing a resolution step on variable x such that v is the first junction point of all the paths starting from C. The existence of this

node is ensured since this junction point is eventually the empty clause. Furthermore, every path starting from the clause C leads to one (and only one) of the premises of the resolution step in the node v. Indeed, a path leading to both premises entails the existence of an intermediate non-read-once clause which is not possible since the refutation is tree-like. We partition the branches starting from C into two subsets containing respectively the paths leading to the first and second premise of the resolution step in the node v. Each partition is non empty since if there exists an empty subset v can't be the first junction point of the branches. Let x be the variable eliminated at this resolution step and suppose w.l.o.g that the first premise contains literal x while the second contains literal $\overline{x}$. As π is regular, x is not a variable of C and the subset of branches starting from C leading to the first premise accepts the substitution of C by $C \vee x$ while the subset of branches leading to the second premise accepts the substitution of C by $C \vee \overline{x}$.

■

The result established in Lemma 6.1 ensures the possibility to fix any non-read-once clause used $k > 1$ times by using the split rule. Indeed, we can apply this rule to replace a non-read-once clause used $k > 1$ times by two clauses used respectively $1 \leq k_1 < k$ and $1 \leq k_2 < k$ such that $k = k_1 + k_2$. By iterating this method, we can fix every non-read-once clause. Then, we only need to replace the resolution rule by the Max-SAT resolution rule to obtain an adaptation from any regular tree resolution refutation to a max-refutation in linear size.

Theorem 6.1. *Given an unsatisfiable formula ϕ and a regular tree resolution refutation π of ϕ, there exists a max-refutation of ϕ containing $O(|\pi|)$ inference steps.*

Proof. Let π be a regular tree resolution refutation of ϕ. We set $T_1 = \emptyset$ and $T_2 = MR(\pi)$, where $MR(\pi)$ is obtained from π after replacing each resolution by Max-SAT resolution. If π is read-once, T_2 is a max-refutation of ϕ containing $|\pi|$ inference steps. Now, let C be a non-read-once clause of π. Using Lemma 6.1, there exists a variable $x \notin var(C)$ and a partition of the branches starting from C into two non-empty subsets, the first accepting $C \vee x$ and the second accepting $C \vee \overline{x}$. We apply the Max-SAT split rule on C to obtain $C \vee x$ and $C \vee \overline{x}$ and we replace C as premise by $C \vee x$ on the first subset of branches and C by $C \vee \overline{x}$ on the second. Doing this, we augment T_1 by adding one split and we change T_2 by replacing the premise clause C as described above. As T_2 is a tree-like regular resolution refutation of $(\phi \backslash C) \wedge (C \vee x) \wedge (C \vee \overline{x})$, it is possible to iteratively apply this operation on T_2 until we obtain a read-once regular tree resolution refutation. Therefore, after the last iteration, we have a pair (T_1, T_2) such that T_1 is a sequence of applications of the split rule transforming ϕ into a Max-SAT equivalent ϕ' and T_2 is a read-once regular max-refutation of ϕ'. Therefore, these transformations form a max-refutation of ϕ.

Next, we prove that the size of the max-refutation is in $O(|\pi|)$. Given a non-read-once leaf clause C used $k > 1$ times, we can easily prove by induction on k that it is possible to fix C using at most $k-1$ splits. Now, let $C_1, ..., C_p$ be the leaf clauses of π used respectively $k_1, k_2, ..., k_p$ times. Notice that $k_1 + k_2 + ... + k_p = |\pi| + 1$ since π has exactly $2|\pi|$ premises and $|\pi| - 1$ intermediate clauses. Using the previous induction, we need at most $k_1 - 1 + k_2 - 1 + ... + k_p - 1 \leq |\pi|$ splits to fix every non-read-once leaf clause of π. Consequently, $|T_1| \leq |\pi|$.

On the other hand, the number of Max-SAT resolutions in T_2 is by construction equal to the number of resolution steps in π and, therefore, $|T_2| = |\pi|$. We conclude that the complete max-refutation contains at most $2|\pi|$ inference steps, which is in $O(|\pi|)$.

■

Example 6.1. *We consider the regular tree resolution refutation of* $\phi = (x_1) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$ *represented in Figure 2.7. We observe that the original clause* (x_1) *is used two times as a premise of a resolution step. The junction point of the left and right branches eliminates variable* x_3 *such that the branch on the left leads to the premise containing literal* x_3 *and the branch on right leads to the premise containing literal* $\overline{x_3}$*. We apply the split rule on clause* (x_1) *to get* $(x_1 \vee x_3)$ *and* $(x_1 \vee \overline{x_3})$ *and we replace* (x_1) *by* $(x_1 \vee x_3)$ *and* $(x_1 \vee \overline{x_3})$ *respectively on the left and right branches. Finally, we replace all resolutions by Max-SAT resolutions to obtain the complete max-refutation represented in Figure 6.1.*

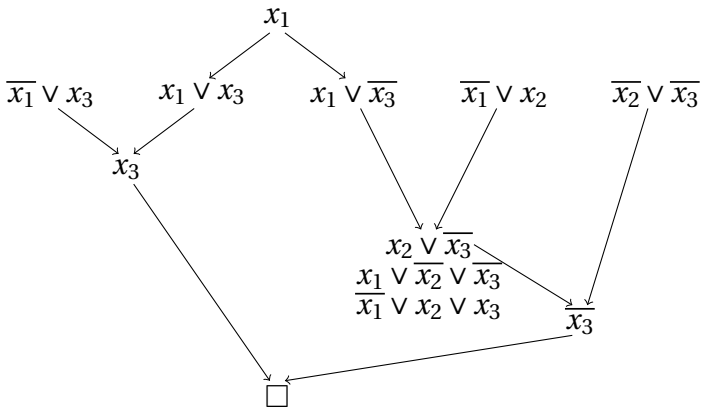


Figure 6.1: From regular tree resolution refutations to linear-size max-refutations

6.2.2 From (Semi-)Tree Resolution Refutations to Max-Refutations

In the previous section, we proposed a linear adaptation from regular tree resolution refutations to max-refutations. We propose to extend the case where this adaptation guarantees linear size of the obtained max-refutation to tree resolution refutations. To this end, we simply rely on the known transformation from any tree resolution refutation to a regular tree resolution refutation established in Lemma 2.1. Since it is possible to make a tree resolution refutation regular without increasing the size of the proof, we can apply the adaptation in Theorem 6.1 to produce a max-refutation with linear size as shown in the following corollary.

Corollary 6.1. *Given an unsatisfiable formula* ϕ *and a tree resolution refutation* π *of* ϕ*, there exists a max-refutation of* ϕ *containing* $O(|\pi|)$ *inference steps.*

To go further in terms of linear adaptations, we propose the class of semi-tree resolution refutations defined below. This class of refutations clearly extends tree resolution refutations, i.e., every tree resolution refutation is semi-tree-like. To extend our linear adaptation

result to semi-tree-like resolution refutations, we propose a method which relies on the simple fact that semi-tree resolution refutations can be partitioned into two parts where the first part is a read-once sequence of resolutions and the second part is a tree-like resolution refutation. As the first part is a read-once sequence of resolutions, it can be trivially adapted into a max-refutation by replacing each resolution by a Max-SAT resolution while the second part can be adapted using the result in Corollary 6.1. After transforming the two parts, we glue them back to construct the complete max-refutation.

Definition 6.1 (Semi-Tree Resolution Refutation)**.** *A resolution refutation is semi-tree-like if, for any branch of the refutation, at most one clause is non-read-once.*

Corollary 6.2. *Given an unsatisfiable formula ϕ and a semi-tree resolution refutation π of ϕ, there exists a max-refutation of ϕ containing $O(|\pi|)$ inference steps.*

Example 6.2. *We consider the resolution refutation π in Figure 6.2. π is clearly semi-tree-like since in each branch at most one clause is non-read-once. Note that π is not tree-like since (x_1) is an intermediate non-read-once clause. To adapt this semi-tree resolution refutation to a max-refutation, we put aside the top resolution on variable x_1 taking clauses $(x_1 \vee x_4)$ and $(x_1 \vee \overline{x_4})$ and we obtain the tree-like resolution refutation represented in Figure 2.7. We adapt this tree-like resolution refutation as in Example 6.1 and we replace the resolution step on x_1 by a Max-SAT resolution step. We glue back the two parts to obtain the complete max-refutation represented in Figure 6.3.*

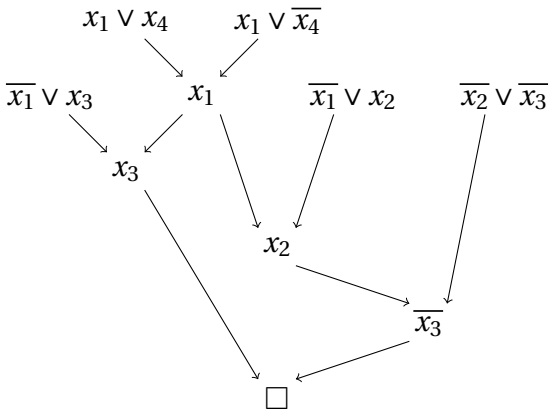


Figure 6.2: Semi-tree-like resolution refutation

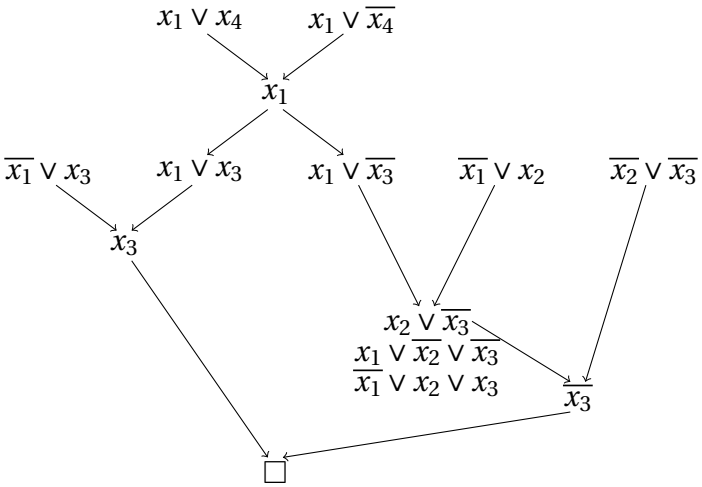


Figure 6.3: Adapting a semi-tree resolution refutation to a max-refutation

6.2.3 From Unrestricted Resolution Refutations to Max-Refutations

In the previous sections, we proposed linear adaptations from specific classes of resolution refutations to max-refutations. These results are established in the following corollary using the notion of simulation. Next, we want to devise an adaptation for the unrestricted case, even with a potential blow-up in the size of the resulting max-refutation.

Corollary 6.3. *ResS p-simulates regular tree, tree and semi-tree resolution.*

To devise our generic adaptation, we simply extend the one described in section 6.2.2 by adding a first transformation to make the initial resolution refutation tree-like. Notice that we could make the initial resolution refutation semi-tree-like (instead of tree-like) but this choice does not affect the theoretical size of the obtained max-refutation. To achieve this first intermediate transformation, we will iteratively search in the proof for the first non-read-once intermediate clause C. If this clause is used $k > 1$ times as a premise of another resolution step, we consider the part of the proof leading to C and we duplicate it k times in order to get a tree-like sequence of resolutions generating k resolvents $C_1, C_2, ..., C_k$, each resolvent C_i containing exactly the same literals as C and is generated by a similar sequence of resolution steps. Consequently, C is no longer used several times as a premise of a resolution step, the input clauses are. Repeating this operation forces the resolution refutation to become tree-like. Fixing a non-read-once intermediate clause can, in the worst case, double the size of the current resolution refutation. As such, the size of the obtained tree-like resolution refutation is exponentially bounded by the size of the initial unrestricted resolution refutation. To polish this upper bound, we introduce a new parameter defined below, which is the number of multi-uses of intermediate clauses. Notice how, in the definition, we subtract 1 use for each clause. Intuitively, we consider the first use of any non-read-once intermediate clause as authorized.

Definition 6.2. *Let π be a resolution refutation. The number of multi-uses of intermediate non-read-once clauses, denoted $\mu(\pi)$, is defined as follows:*

$$\mu(\pi) = \sum_{C \text{ intermediate non-read-once in } \pi} (d^+(C) - 1)$$

where $d^+(C)$ denotes the number of uses of the clause C, i.e., the number of outgoing arcs from C in the DAG representation of π.

Theorem 6.2. *Given an unsatisfiable formula ϕ and an unrestricted resolution refutation π of ϕ, there exists a max-refutation of ϕ with $O(2^{\mu(\pi)} \times |\pi|)$ inference steps.*

Proof. Let π be a resolution refutation of ϕ. We iteratively make the intermediate non-read-once clauses read-once. Each time, we pick the first intermediate non-read-once clause C and duplicate the sub-proof deriving C exactly $d^+(C) - 1$ times. Each iteration decrements the number of intermediate non-read-once clauses by 1 until the resolution refutation becomes tree-like. Clearly, for each duplication, the size of the proof is doubled in the worst case and we perform exactly $\mu(\pi)$ duplications. The size of the obtained tree resolution refutation π' is thus bounded by $O(2^{\mu(\pi)} \times |\pi|)$. Then, using Theorem 6.1, we obtain a max-refutation of size $O(2^{\mu(\pi)} \times |\pi|)$.

■

Example 6.3. *We consider the resolution refutation represented in Figure 6.4. This refutation is not semi-tree-like since the clauses (x_1) and (x_4) are two non read-once clauses in the same branch. First, we duplicate the resolutions leading to (x_1) and we obtain the tree-like resolution refutation represented in Figure 6.5. Then, we apply the transformations described in Corollary 6.1 to get the max-refutation represented in Figure 6.6.*

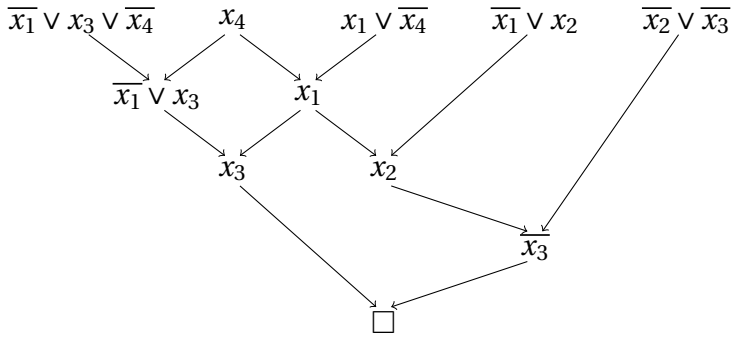


Figure 6.4: Unrestricted resolution refutation

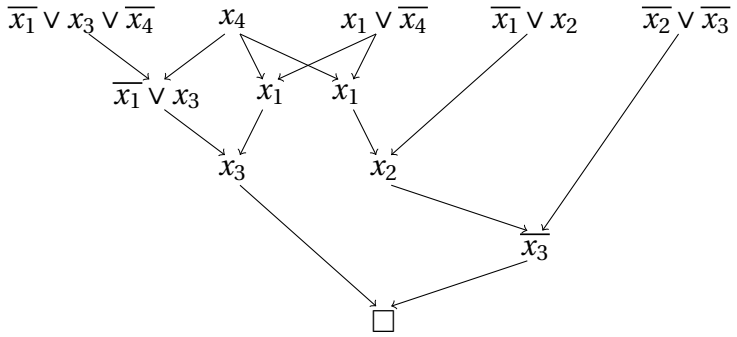


Figure 6.5: Adapting a resolution refutation to a tree-like resolution refutation

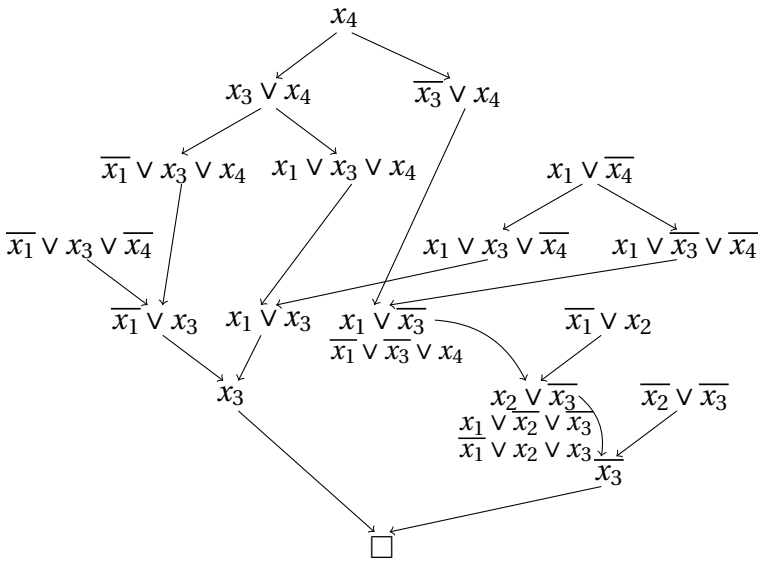


Figure 6.6: Adapting an unrestricted resolution refutation to a max-refutation

6.3 From SAT Refutations to Max-SAT Resolution Refutations

In the previous section, we contributed to the open problem of adapting resolution refutations into Max-SAT resolution refutations. However, we circumvented the problem by augmenting Max-SAT resolution with the split rule thus generating max-refutations in the ResS system [LR20b]. In this section, we tackle the original and more difficult problem of adapting resolution proofs into proofs valid for Max-SAT using exclusively Max-SAT resolution. More specifically, we identify a new fragment of resolution, called crossing-free resolution, for which an efficient adaptation using only Max-SAT resolution is possible. Crossing-free derivations are defined using the ensuing derivations of non read-once clauses. Intuitively, non read-once clauses are used independently to infer new information

in crossing-free resolution proofs. The adaptation of such proofs to Max-SAT resolution proofs is shown possible modulo some minor syntactic subtleties.

6.3.1 Crossing-Free Resolution

The main difficulty in adapting resolution proofs to Max-SAT resolution ones lies in inferring a substitute for non read-once clauses. Indeed, such clauses must be naturally inferred using Max-SAT resolution while unfolding (i.e., reading and applying) the initial resolution proof, contrary to the previous adaptations in Section 6.2 where non read-once clauses are artificially fixed using the split rule before the actual unfolding of the proof. In this section, we define a new fragment of resolution, referred to as crossing-free resolution. The idea behind this refinement is to ensure enough maneuverability of proofs in terms of structure in order to infer substitutes for non read-once clauses when necessary. To this end, we define below the notion of ensuing derivation of a non read-once clause. Intuitively, this particular derivation is ensued from a non read-once clause in the sense that it is sufficient to delimit the impact of its multiple uses. We recall that a node where a set of given paths in a resolution proof intersect is referred to as their junction node.

Definition 6.3 (Ensuing Derivation)**.** *Let ϕ be a CNF formula and π a resolution derivation of clause C from ϕ. The ensuing derivation of a non read-once clause C' in π, denoted $ED(C')$, is the sub-derivation of π formed by all the resolution steps in the paths starting from C' in π until their first junction node. We call the clause derived in the junction node, the ensued clause of C', denoted $EC(C')$.*

Example 6.4. *We consider the resolution derivation π represented in Figure 6.7 of clause $C = x_6$ from the formula $\phi = \{\overline{x_1} \vee x_3 \vee \overline{x_4},\ x_4 \vee x_5,\ x_4 \vee \overline{x_5},\ x_1 \vee \overline{x_4},\ x_2 \vee \overline{x_4} \vee x_6,\ x_5 \vee \overline{x_7},\ \overline{x_2} \vee \overline{x_3} \vee x_7,\ \overline{x_5} \vee \overline{x_7}\}$. The non read-once clauses x_4 and $\overline{x_2} \vee \overline{x_3} \vee x_7$ and their ensuing derivations are respectively represented in red and blue. Furthermore, we have $EC(x_4) = x_6$ and $EC(\overline{x_2} \vee \overline{x_3} \vee x_7) = \overline{x_2} \vee \overline{x_3}$.*

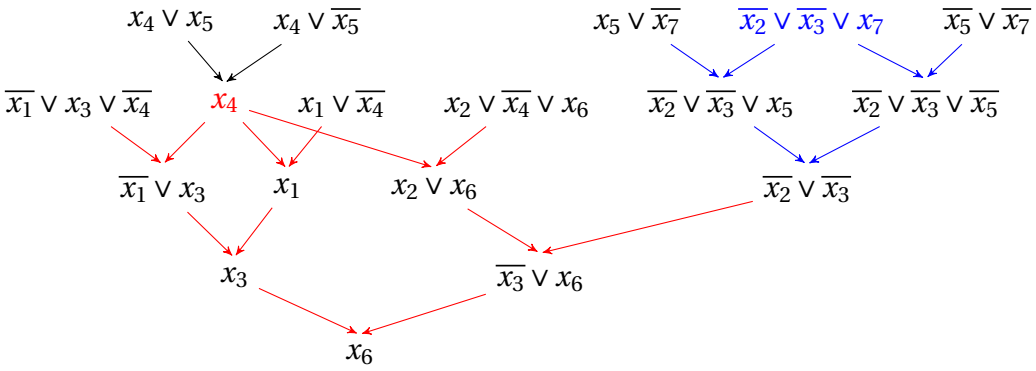


Figure 6.7: Ensuing derivations in a crossing-free resolution proof

Recall that clauses are consumed after the application of Max-SAT resolution. Therefore, it seems difficult to adapt resolution derivations in which ensuing derivations of non

read-once clauses cross. Indeed, in such cases, the formula can significantly evolve as compensation clauses may be used while others may be generated. As such, crossing-free resolution ensures that ensuing derivations are disjoint, i.e., do not cross, as defined below.

Definition 6.4 (Crossing-Free Resolution)**.** *Let ϕ be a CNF formula and π a resolution derivation of clause C from ϕ. π is crossing-free if for every pair of non read-once clauses (C_1, C_2), $ED(C_1)$ and $ED(C_2)$ are disjoint, i.e., they do not contain a shared arc.*

Example 6.5. *We consider the same formula ϕ in Example 6.4. The resolution derivation π of clause $C = x_6$ from ϕ represented in Figure 6.7 is crossing-free since the ensuing derivations of the non read-once clauses x_4 and $\overline{x_2} \vee \overline{x_3} \vee x_7$ are disjoint.*

Note that the crossing-free resolution refinement entails an interesting property established in the following proposition. Intuitively, this property ensures that non read-once clauses are used independently to infer new information in crossing-free resolution proofs. This entails that each ensuing derivation in a crossing-free resolution proof can be adapted independently as described in the next section.

Proposition 6.1. *Let ϕ be a CNF formula, π be a crossing-free resolution derivation of clause C from ϕ and C' a non read-once clause in π. Every clause Cl in $ED(C')$ s.t. $Cl \notin \{C', EC(C')\}$ is read-once.*

Proof. Let Cl be a clause in $ED(C')$ s.t. $Cl \notin \{C', EC(C')\}$. Clearly, if Cl is not read once, $ED(Cl)$ shares at least one arc with $ED(C')$ which is absurd since π is crossing-free. ■

Note that in the following section where we will introduce our adaptation from crossing-free proofs into Max-SAT resolution proofs, we may allow the addition of tautological clauses to any formula in the proof. Furthermore, for the sake of simplification, we will temporarily allow the use of the following rewriting $C \vee \overline{a_1 \vee a_2 \vee ... \vee a_n} = (C \vee \overline{a_1}) \wedge (C \vee a_1 \vee \overline{a_2}) \wedge ... \wedge (C \vee a_1 \vee a_2 \vee ... \vee \overline{a_n})$ [3] [LHG08] as two full-fledged rules to manipulate clauses in compacted form. We will refer to the left-right rewriting as expansion and to the right-left one as compaction. This may entail abusing some notations in the following section and we thoroughly discuss these syntactic subtitles in Section 6.3.3.

6.3.2 From Crossing-Free Resolution to Max-SAT Resolution

In this section, we show that crossing-free resolution derivations can be adapted to Max-SAT resolution derivations modulo some minor syntactic subtleties without substantially increasing their size. In the following proposition, we first provide some patterns which will be encountered in the adaptation.

Proposition 6.2. *Let A, B, C and $\{l\}$ be four sets of literals s.t. $|C| > 0$. The following deductions can be done in $O(|C|)$ inference steps:*

[3] refer to Section 3.3.2

(a) $(A \vee C) \wedge (B \vee \overline{C}) \vdash_{MaxRes} A \vee B$

(b) $(l \vee A \vee \overline{C}) \wedge (\overline{l} \vee B) \vdash_{MaxRes} A \vee B \vee \overline{C}$

(c) $(l \vee A \vee \overline{C}) \wedge (\overline{l} \vee B \vee \overline{C}) \vdash_{MaxRes} A \vee B \vee \overline{C}$

Proof. We provide the proof for case (a) by induction on $|C| = n$:

- If $n = 1$, then $C = \{l'\}$. Clearly, $(A \vee l') \wedge (B \vee \overline{l'}) \vdash_{MaxRes} A \vee B$ by application of a Max-SAT resolution step on literal $var(l')$.
- Suppose $n > 1$ and let $l' \in C$. By the induction hypothesis, we can deduce $(A \vee C) \wedge (B \vee \overline{C \setminus \{l'\}}) \vdash_{MaxRes} A \vee B \vee l'$ in $n-1$ inference steps. Furthermore, $B \vee \overline{C} = (B \vee \overline{C \setminus \{l'\}}) \wedge (B \vee \overline{l'})$ by expansion and $(A \vee B \vee l') \wedge (B \vee \overline{l'}) \vdash_{MaxRes} A \vee B$ by application of a Max-SAT resolution step on variable $var(l')$. Therefore, we conclude that we can deduce $(A \vee C) \wedge (B \vee \overline{C})\} \vdash_{MaxRes} A \vee B$ in $O(n)$ inference steps.

Proofs for cases (b) and (c) are similar by induction on $|C|$.

■

Next, we start dealing with the adaptation of crossing-free resolution derivations and particularly ensuing derivations. To generate a substitute for a non read-once clause, note that we can use the literals in the junction nodes of an ensuing derivation, i.e., the nodes where paths starting from the non read-once clause intersect. To generate such substitutes using Max-SAT resolution, we start by dealing with read-once linear parts in the proof. Informally, we want to drag (i.e., bring along) each non read-once clause while unfolding the proof until they are reused. This is formally established for read-once linear parts of the proof in the following lemma. Note that the implications of equality $\stackrel{*}{=}$ in the proof will be further discussed in Section 6.3.3.

Lemma 6.2. *Let ϕ be a CNF formula, $\pi = C_1, ..., C_{s(\pi)}$ be a read-once linear resolution derivation of clause $C \neq \square$ from ϕ. We can deduce $\phi \vdash_{MaxRes} C \wedge (C_1 \vee \overline{C})$ in $O(s(\pi) \times w(\pi))$ inference steps.*

Proof. Let $m = s(\pi)$. Since π is read-once, it can be trivially adapted into a Max-SAT resolution derivation of C from ϕ of the same size by replacing every resolution step with a Max-SAT resolution step. Next, we prove by induction on $i \in \{1, .., m-1\}$ that we can infer $C'_i = C_1 \vee \overline{C_{i+1}}$ at the i^{th} Max-SAT resolution step:

- For $i = 1$, the first Max-SAT resolution on clauses $C_1 = l_1 \vee A_1$ and $D_1 = \overline{l_1} \vee B_1$ w.r.t $var(l_1)$ generates the following compensation clause:

$$CC_{1|1} = l_1 \vee A_1 \vee \overline{B_1} \stackrel{*}{=} l_1 \vee A_1 \vee \overline{A_1 \vee B_1} = C_1 \vee \overline{C_2} = C'_1$$

  Note that to establish the equality $\stackrel{*}{=}$ [4], we can add the tautological clauses $l_1 \vee A_1 \vee B_1 \vee \overline{A_1}$ (or alternatively $l_1 \vee A_1 \vee \overline{A_1}$) to the formula) in which case $l_1 \vee A_1 \vee \overline{A_1 \vee B_1}$

[4] which is sound for Max-SAT (c.f. Remark 13 in [LHG08])

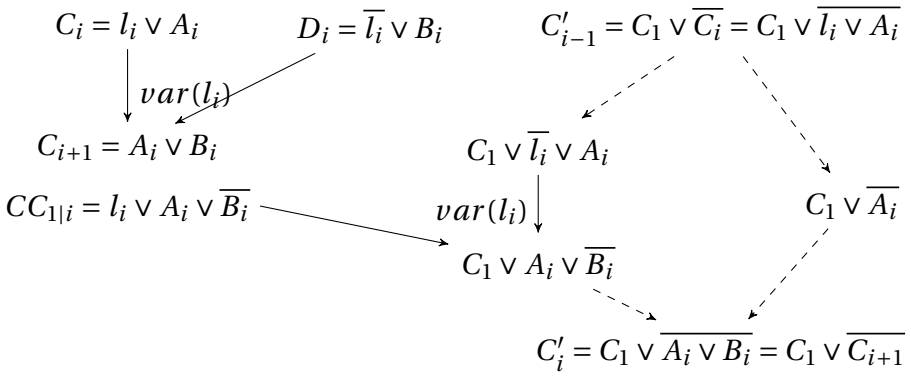


Figure 6.8: Induction step to infer C'_i at the i^{th} step. Solid lines represent the application of the Max-SAT resolution rule whereas dashed lines represent compaction or expansion. Unused compensation clauses are omitted.

can be trivially inferred by compaction. Furthermore, if D_1 is a unit clause, $CC_{1|1}$ is not generated. However, we can simply add the tautological clauses $l_1 \vee A_1 \vee \overline{A_1}$ which correspond to $C_1 \vee \overline{C_2}$ since $D_1 = \overline{l_1}$ (i.e., B_1 is empty).

- Suppose that we can generate $C'_{i-1} = C_1 \vee \overline{C_i}$ at the $i^{th} - 1$ Max-SAT resolution step. The i^{th} step on $C_i = l_i \vee A_i$ and $D_i = \overline{l_i} \vee B_i$ w.r.t $var(l_i)$ generates the resolvent $C_{i+1} = A_i \vee B_i$ and the compensation clauses $CC_{1|i} = l_i \vee A_i \vee \overline{B_i}$ and $CC_{2|i} = \overline{l_i} \vee \overline{A_i} \vee B$. The induction step to infer C'_i is represented in Figure 6.8. Note that similarly to the base case, if $D_i = \overline{l_i} (i > 1)$ is a unit clause, i.e., the i^{th} step corresponds to a deletion of literal l_i from $C_i = l_i \vee A_i$ deducing the resolvent $C_{i+1} = A_i$, the tautological clauses $l_i \vee A_i \vee \overline{A_i}$ can be added to the formula thus replacing $CC_{1|i}$ in Figure 6.8. However, as showcased in the same figure, the addition of such clauses in case D_1 is unit can be avoided since the initial expansion step on C'_{i-1} suffices to generate $C_1 \vee \overline{A_i} = C_1 \vee \overline{C_{i+1}} = C'_i$.

Finally, by Proposition 6.2 (case b.), the inference of $C_1 \vee A_i \vee \overline{B_i}$ in Figure 6.8 requires $O(|B_i|)$ Max-SAT resolution steps and, thus, every step in π is clearly adapted in $O(w(\pi))$ inference steps to generate C and $C_1 \vee \overline{C_m}$. Therefore, we conclude that we can deduce $\phi \vdash_{MaxRes} C \wedge (C_1 \vee \overline{C_m})$ in $O(s(\pi).w(\pi))$ inference steps.

■

Example 6.6. *We consider the read-once linear derivation of clause* $\overline{x_3} \vee x_6$ *from* $\phi = \{x_4,\ x_2 \vee \overline{x_4} \vee x_6,\ \overline{x_2} \vee \overline{x_3}\}$ *represented on the left of Figure 6.9. The Max-SAT resolution proof deducing* $\overline{x_3} \vee x_6$ *and* $x_4 \vee \overline{x_3} \vee x_6$ *is represented on the right of Figure 6.9.*

Next, we establish our main result on the adaptation of crossing-free resolution derivations. The proof in the following theorem particularly deals with the junction nodes in

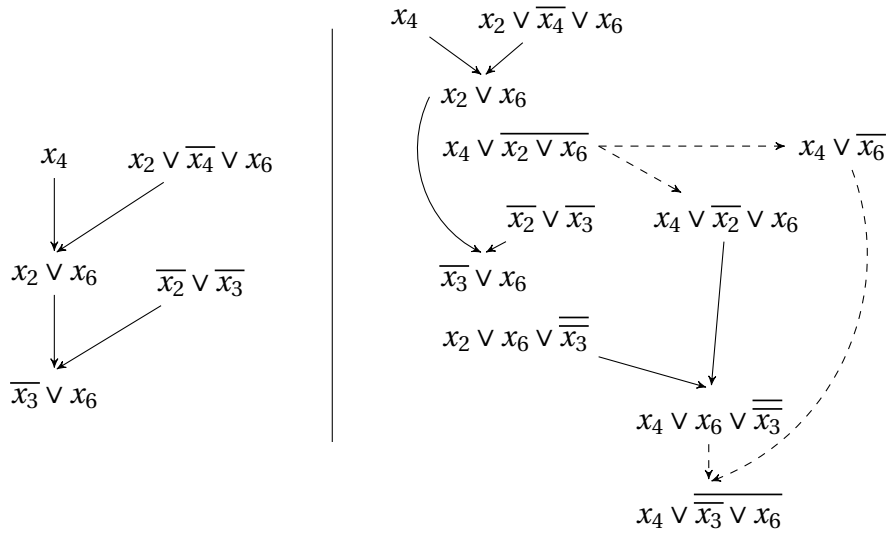


Figure 6.9: Dragging the non read-once clause while unfolding a read-once linear section of the proof. Solid lines represent the application of the Max-SAT resolution rule whereas dashed lines represent compaction or expansion. Unused compensation clauses are omitted.

ensuing derivations, i.e., nodes where the paths starting from the non read-once clauses intersect. More specifically, we want to drag or bring along the non read-once clause through these particular nodes. We provide an illustration of a full adaptation in Example 6.7.

Theorem 6.3. *Let ϕ be a CNF formula and π be a crossing-free resolution derivation of clause C from ϕ. We can deduce $\phi \vdash_{MaxRes} C$ in $O(s(\pi) \times (s(\pi) + w(\pi))^2)$ inference steps.*

Proof. Property 6.1 ensures that each ensuing derivation can be adapted independently. Let Cl be a non read-once clause in π and w.l.o.g we only consider it ensuing derivation $ED(Cl)$. We prove that at each step of $ED(Cl)$ deriving clause C', we can infer C' and $SC \vee \overline{C'}$ where SC is either Cl or its substitute in the path leading to C'. The proof is by induction on the size of the derivation. The base case where the derivation is empty is trivial. Next, using Lemma 6.2, we can suppose w.l.o.g that C' is derived in a junction node (of paths starting from Cl). Let $l \vee A$ and $\overline{l} \vee B$ be the premises of the resolution step deriving $C' = A \vee B$.

The induction hypothesis ensures that there exists a Max-SAT resolution derivation of $l \vee A$ and $C \vee \overline{l \vee A}$. As showcased in Figure 6.10, $Cl \vee \overline{l}$ can be used to replace the occurrences of Cl in the derivation of $\overline{l} \vee B$. Note that to avoid using tautological substitutes, we can suppose w.l.o.g that $l \notin Cl$ by interchanging the proofs of $l \vee A$ and $l \vee B$ when necessary thus entailing a different unfolding order of the original proof and the generation of the exact same clause as a substitute in such nodes. Again, similarly to the left side, the induction hypothesis ensures the existence of a Max-SAT resolution derivation of $\overline{l} \vee B$ and $Cl \vee \overline{l} \vee \overline{\overline{l} \vee B}$ and, therefore, $Cl \vee \overline{l} \vee \overline{B}$ by expansion. Clearly, $C' = A \vee B$ can be derived by Max-SAT resolution and we showcase in Figure 6.10 how $Cl \vee \overline{C'} = C \vee \overline{A \vee B}$ can be inferred using the compensation clauses as well as $Cl \vee \overline{l \vee A}$ and $Cl \vee \overline{l} \vee \overline{B}$.

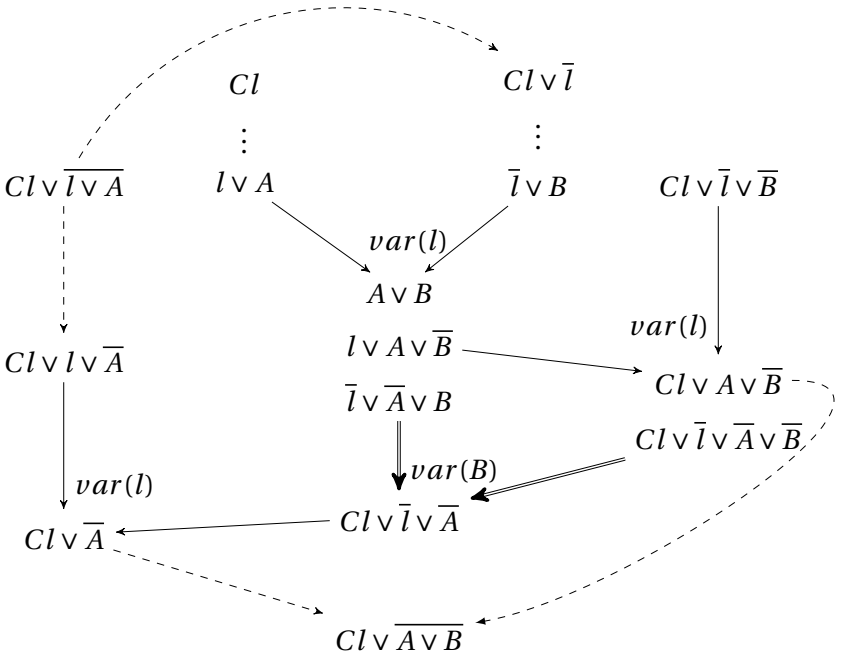


Figure 6.10: Inferring $Cl \vee \overline{A \vee B}$ in a junction node of $ED(Cl)$. Solid lines represent the application of the Max-SAT resolution rule, bold double arcs represent the application of Max-SAT resolution to delete opposed sets of literals and dashed lines represent compaction or expansion. Unused compensation clauses are omitted.

Note that the following particular cases can occur:

- A or B is empty, in which case a unit clause is used to derive $C' = A \vee B$. We represent in Figure 6.11 how to derive $Cl \vee \overline{C'}$ in case A is empty. The derivation in case B is empty is symmetric and thus omitted. Notice that in the case both A and B are empty, π is a refutation and there is no need to derive $Cl \vee \overline{C'}$ in the last Max-SAT resolution step. In fact, more generally, this is also not necessary for the last junction node in an ensuing derivation in π.
- $A = B$ in which case the generated compensation clauses are tautological and are not necessary to derive $Cl \vee \overline{C'} = Cl \vee \overline{A}$ as showcased in Figure 6.12.

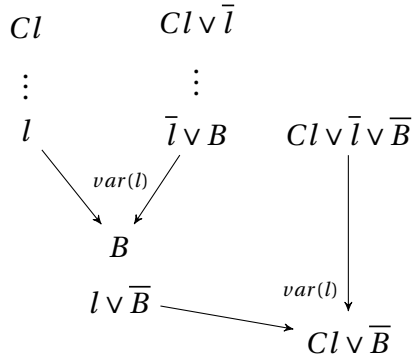


Figure 6.11: Inferring $Cl \vee \overline{C'}$ in case A is empty in a junction node of $ED(Cl)$. Solid lines represent the application of the Max-SAT resolution rule whereas dashed lines represent compaction and expansion. Unused compensation clauses are omitted.

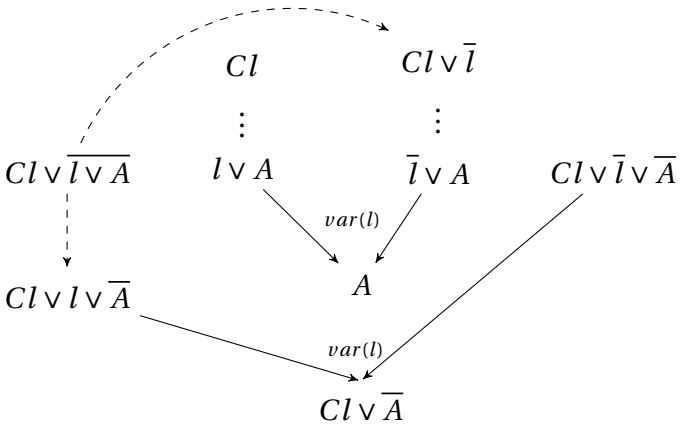


Figure 6.12: Inferring $Cl \vee \overline{C'}$ in case $A = B$ in a junction node of $ED(Cl)$. Solid lines represent the application of the Max-SAT resolution rule whereas dashed lines represent compaction and expansion. Unused compensation clauses are omitted.

Finally, in each junction node we need $O(|B|)$ inference steps to deduce $Cl \vee A \vee \overline{B}$ using case (c) in Proposition 6.2. Similarly, using expansion on A and pattern (b) in Proposition 6.2, we need $O(|A| \times |B|)$ inference steps to deduce $Cl \vee \overline{l} \vee \overline{A}$. It is important to note that the width of the proof may evolve while generating substitutes for non read-once clauses as literals may be added in junction nodes. However, the width remains bounded by $w(\pi) + s(\pi)$ and thus each junction node can be adapted in $O((w(\pi) + s(\pi))^2)$ inference steps. Therefore, we conclude that we can deduce $\phi \vdash_{MaxRes} C$ in $O(s(\pi) \times (s(\pi) + w(\pi))^2)$ inference steps. ■

Example 6.7. *We consider the formula* $\phi = \{\overline{x_1} \vee x_3 \vee \overline{x_4},\ x_4 \vee x_5,\ x_4 \vee \overline{x_5},\ x_1 \vee \overline{x_4},\ x_2 \vee \overline{x_4} \vee x_6,\ \overline{x_2} \vee \overline{x_3}\}$ *and the derivation* π *of clause* x_6 *from* ϕ *represented in Figure 6.7. We omit the*

section of the proof (in blue) deriving clause $\overline{x_2} \vee \overline{x_3}$ for simplicity. Note that this omitted part, i.e., the ensuing derivation of the non read-once clause $\overline{x_2} \vee \overline{x_3} \vee \overline{x_7}$ corresponds to a diamond pattern. Such patterns will be studied in Section 6.4. The adaptation of proof π is reported in Figure 6.13. We reuse the adaptation of the linear read-once section in Example 6.9. The non read-once clause and its substitutes are colored in red and added tautological clauses are represented in green. Note that this is one of the possible adaptations depending on the order chosen for adapting the branches of $ED(x_4)$. Finally, we stress the fact that we could have generated the clause $C = x_4 \vee \overline{x_6}$ after the last Max-SAT resolution step on clauses $\overline{x_3} \vee x_3$ (but we omit this inference since $x_6 = EC(x_4)$ as mentioned in the proof of Theorem 6.3). Indeed, C can be inferred by an additional Max-SAT resolution step on the compensation clauses obtained in the last step, i.e., clauses $x_3 \vee \overline{x_6}$ and $x_4 \vee \overline{x_3} \vee \overline{x_6}$. In the proof of Theorem 6.3, this corresponds to the case where B is empty in a junction node of an ensuing derivation.

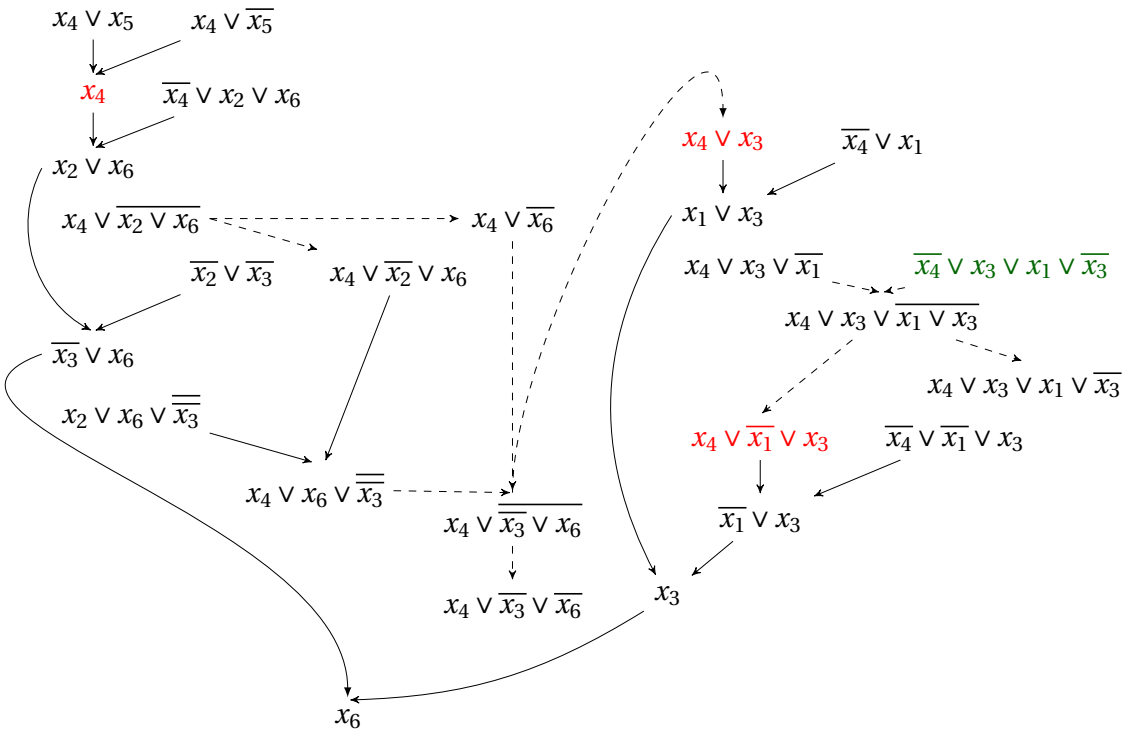


Figure 6.13: Adaptation of a crossing-free resolution derivation. Solid lines represent the application of the Max-SAT resolution rule whereas dashed lines represent compaction and expansion. Unused compensation clauses are omitted.

6.3.3 On Orderings and Tautological Clauses

In this section, we discuss some minor syntactic subtleties that occur in the adaptation. First, it is important to note that the use of the expansion and compaction rewritings as full fledged rules is relevant for simplification but not necessary. Recall that these two rules

are mainly used in order to switch between the different equivalent forms of $\overline{C}$ when it is written in CNF form. Each form corresponds to a different ordering of the literals in C. When applying Max-SAT resolution, a relevant order may be chosen when necessary. However, an application of a compaction followed by an expansion may correspond to a certain rearrangement of the variables in CNF form. This may occur when adapting the read-once linear part of the proof. Indeed, as showcased in Figure 6.8, a compaction may be followed by an expansion to isolate the clause $C_1 \vee \overline{l_i} \vee A_i$ from the compact form $C_1 \vee \overline{A_i}$. Similarly, as shown in Figure 6.10, it may be necessary to isolate the clause $Cl \vee \overline{l}$ from the compact form $Cl \vee l \vee \overline{A}$ when dealing with junction nodes.

More specifically, we may need to rearrange a certain literal at the beginning or at the end of the ordering. In Proposition 6.3, we prove that it is possible to switch the first and last literals in the CNF form of $\overline{C}$ in $O(|C|)$ inference steps. This entails that in the proof of Theorem 6.3, the compaction and expansion rules can be omitted and replaced with $O(s(\pi) \times (s(\pi) + w(\pi)))$ Max-SAT resolutions. Clearly, this does not impact our result in terms of the size of the resulting adaptation. In Example 6.8, we provide the full simplified adaptation of the proof in Example 6.7 without the use of rewriting rules.

Proposition 6.3. *Let n be a natural number and $l_1, ..., l_n$ be n literals. We can deduce $(\overline{l_1}) \wedge (l_1 \vee \overline{l_2}) \wedge ... \wedge (l_1 \vee ... \vee l_{n-1} \vee \overline{l_n}) \vdash_{MaxRes} (\overline{l_n}) \wedge (l_n \vee \overline{l_2}) \wedge ... \wedge (l_n \vee l_2 \vee ... \vee l_{n-1} \vee \overline{l_1})$ in $O(n)$ inference steps.*

Proof. By induction on n we have:

- If $n = 1$ the result is trivial.
- For $n > 1$, the application of Max-SAT resolution on clauses $l_1 \vee .... \vee l_{n-2} \vee \overline{l_{n-1}}$ and $l_1 \vee .... \vee l_{n-1} \vee \overline{l_n}$ w.r.t $var(l_{n-1})$ generates the resolvent clause $C = l_1 \vee .... \vee l_{n-2} \vee \overline{l_n}$ and the compensation clause $CC = l_1 \vee .... \vee l_{n-2} \vee l_n \vee \overline{l_{n-1}}$. Furthermore, by induction, we can deduce $(\overline{l_1}) \wedge (l_1 \vee \overline{l_2}) \wedge ... \wedge (l_1 \vee ... \vee l_{n-2} \vee \overline{l_n}) \vdash_{MaxRes} (\overline{l_n}) \wedge (l_n \vee \overline{l_2}) \wedge ... \wedge (l_n \vee l_2 \vee ... \vee l_{n-2} \vee \overline{l_1})$ in $O(n-1)$ inference steps. A single additional Max-SAT resolution step on clauses CC and $l_n \vee l_2 \vee ... \vee l_{n-2} \vee \overline{l_1}$ w.r.t $var(l_1)$ is sufficient to generate the resolvent clause $l_n \vee l_2 \vee ... \vee l_{n-2} \vee \overline{l_{n-1}}$ and the compensation clause $l_n \vee l_2 \vee ... \vee l_{n-1} \vee \overline{l_1}$. Therefore, we deduce the wanted result in $O(n)$ inference steps.

■

Example 6.8. *We consider the same formula ϕ in Example 6.7. We represent in Figure 6.14 a Max-SAT resolution proof (without rewriting) of clause x_6 from ϕ. Notice how we use the following rearrangement $\overline{x_6} \wedge (x_6 \vee x_3) \vdash_{MaxRes} x_3 \wedge (x_3 \vee x_6)$ to generate the substitute $x_4 \vee x_3$. Furthermore, the tautological clause $\overline{x_4} \vee x_3 \vee x_1 \vee \overline{x_3}$ colored in green in Figure 6.13 and the rearrangement in which it is involved are not necessary since the last required substitute for x_1, i.e $x_4 \vee \overline{x_1} \vee x_3$ is naturally generated by the preceding Max-SAT resolution step. Therefore, they can be deleted as is the case for the full adaptation without rewriting in Figure 6.14.*

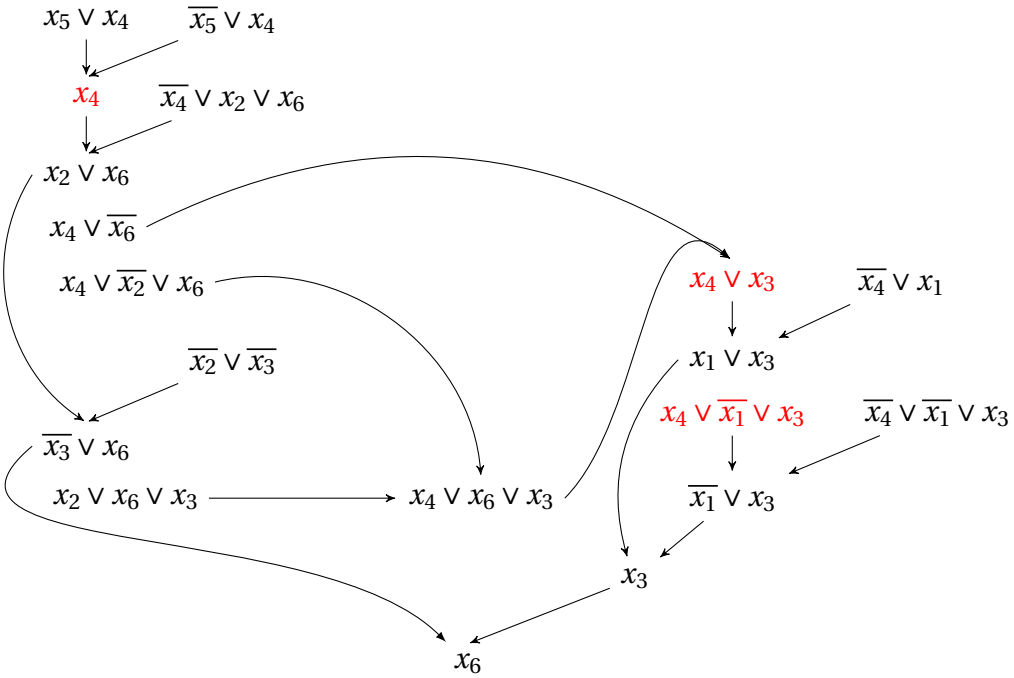


Figure 6.14: Adaptation of a crossing-free resolution proof to a Max-SAT resolution proof. Unused compensation clauses are omitted.

Next, we discuss the implications of the equality $\overset{*}{=}$ used in the proof of Lemma 6.2, i.e., $l \vee A \vee \overline{B} \overset{*}{=} l \vee A \vee \overline{A \vee B}$. This equality is sound for Max-SAT [5]. However, to avoid adding it as a standalone rule and as explained in the proof of Lemma 6.2, we can consider the addition of tautological clauses. This may also be required in case of unit clauses. It is important to note that the number of tautological clauses added to the formula in an adaptation of a crossing-free resolution derivation π is in $O(s(\pi) \times (w(\pi) + s(\pi)))$. A similar phenomenon was also noted in [BP07a] for general and linear resolution [6]. In addition, notice how the adaptation may also rely on tautological compensation clauses which are generated by Max-SAT resolution. Such clauses are usually deleted or omitted in the literature [LH05a; BLM06; BLM07; LHG08] but they may carry important information which is necessary to infer substitutes for non read-once clauses.

6.3.4 From Crossing-Free Resolution Refutations to Max-SAT Resolution Refutations

In the previous sections, we established our result on any resolution derivation, i.e., not necessarily deducing the empty clause. In the following corollary, we project our results on

[5] c.f. Remark 13 in [LHG08]

[6] We similarly avoid the p-simulation terminology in this case although the definition can be relaxed to allow such a subtlety.

the specific case of crossing-free refutations. We also illustrate in Example 6.9 an adaptation of a crossing free resolution refutation into a Max-SAT resolution refutation.

Corollary 6.4. *Let ϕ be an unsatisfiable CNF formula and π be a crossing-free resolution refutation of ϕ. We can deduce $\phi \vdash_{MaxRes} \square$ from ϕ in $O(s(\pi)^3)$ inference steps.*

Proof. Trivially entailed from Theorem 6.3 since $w(\pi) = O(s(\pi))$ for refutations. ■

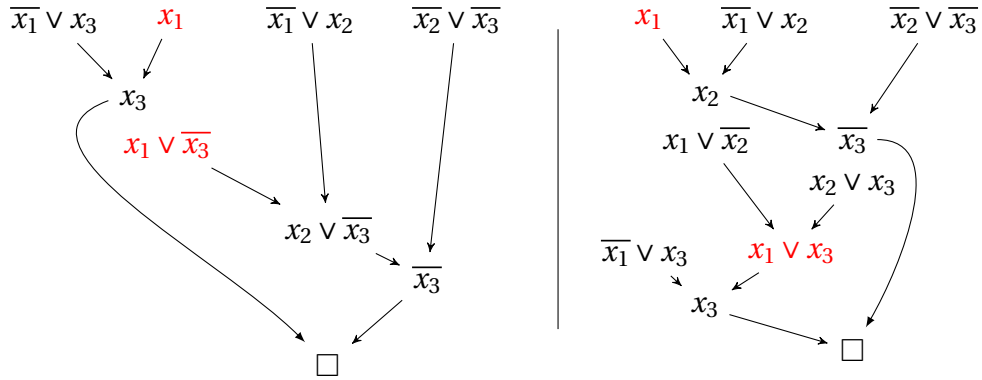


Figure 6.15: Two possible adaptations of the crossing-free resolution refutation represented in Figure 2.7 depending on the ordering of the resolution steps involving the non-read-once clause x_1. Unused compensation clauses are omitted.

Example 6.9. *We consider the unsatisfiable CNF formula $\phi = (x_1) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$ and the refutation π of ϕ represented in Figure 2.7. Clearly, π is crossing-free since there is only one non read-once clause, i.e., x_1. In fact, π also corresponds to the ensuing derivation of x_1 and $\square$ is its ensued clause, i.e., $ED(x_1) = \pi$ and $EC(x_1) = \square$. Two possible adaptations of π are illustrated in Figure 6.15. The non read-once clause and its substitutes are colored in red. The possible adaptations correspond to different possible orderings of the proof. In the adaptation on the left, we consider that the resolution step on clauses $\overline{x_1} \vee x_1$ and x_1 precedes the one on clauses x_1 and $\overline{x_1} \vee x_2$, and inversely for the adaptation on the right. Note that the adaptation on the left corresponds to the handmade example provided by Bonet et al. in [BLM06; BLM07] [7] and thus provides a theoretical understanding of why this handmade proof is valid through the lens of crossing-free resolution.*

6.4 On Diamond Patterns

In this section, we study specific resolution refutations in order to determine the limits and/or features of the proposed adaptations in previous sections. In particular, we show that the adaptation proposed for unrestricted resolution in Corollary 6.2 has a worst case exponential blow-up in the size of the refutations. First, we introduce in the following

[7] c.f. Example 1 in [BLM06] or Example 3 in [BLM07]

definition a new pattern which we will use to build such refutations. We can represent this pattern by a diamond as in Figure 6.17. Notice that the particular diamond pattern $(x, y, \square)$ is a resolution refutation. Furthermore, each diamond pattern π can be trivially adapted into a Max-SAT resolution proof of size $|\pi| - 1$ as shown in Figure 6.18. Our aim is thus to simply use these patterns to theoretically evaluate and compare the proposed adaptations regarding specific resolution refutations.

Definition 6.5 (Diamond)**.** *Let A be a disjunction of literals and let $x \notin var(A)$ and $y \notin var(A)$ two distinct variables. We define the diamond pattern (x, y, A) as the sequence of resolutions represented in Figure 6.16.*

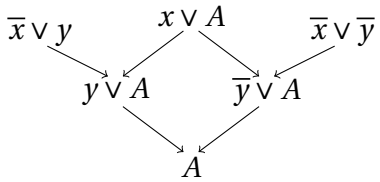


Figure 6.16: Diamond pattern (x, y, A)

Figure 6.17: Simplified representation of a diamond pattern

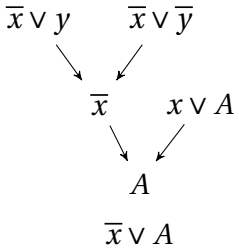


Figure 6.18: Trivial adaptation of a diamond pattern (x, y, A)

Now, imagine that the topmost clause of a diamond pattern $(x, y, \square)$ is derived through another diamond pattern. We iterate the same reasoning to define below a k-stacked diamond pattern. Such a pattern can be represented as a stack of diamonds as shown in Figure 6.19 for $k = 3$. Clearly, k-stacked diamonds are resolution refutations as they deduce the empty clause $\square$.

Definition 6.6 (k-Stacked Diamond)**.** *Let $k \geq 1$ be a natural number and let x_i and y_i where $1 \leq i \leq k$ be distinct variables. A k-stacked diamond pattern is formed by k diamond patterns (x_i, y_i, A_i) where $1 \leq i \leq k$ as follows:*

- $A_1 = \square$ *and* $A_i = (x_1 \vee \cdots \vee x_{i-1})$ *for* $1 < i \leq k$
- *each diamond* (x_i, y_i, A_i) *is stacked on top of* $(x_{i-1}, y_{i-1}, A_{i-1})$ *s.t. the last conclusion of the former is the topmost central premise of the latter*

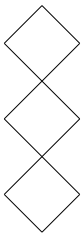


Figure 6.19: Simplified representation of a 3-stacked diamond pattern

Note that, when $k > 2$, a k-stacked diamond is not semi-tree-like. The size of a k-stacked diamond π is $|\pi| = 3k$. Furthermore, we have $\mu(\pi) = k-1$. Therefore, after the application of the adaptation described in Theorem 6.2, we obtain a max-refutation whose size is at least 2^{k-1} showing that the proposed adaptation for the unrestricted case can be exponential in the worst case. However, it is worth noting that if each diamond $\pi_d = (x, y, A)$ is considered independently, we can simply add a single split step on $x \vee A$ with respect to y and replace each resolution with a Max-SAT resolution to obtain a valid Max-SAT derivation of A of size $|\pi_d| + 1$ as showcased in Figure 6.20. This is possible since each diamond can be considered as a regular tree resolution derivation of A in the spirit of Theorem 6.1. Finally, noticing that (k-stacked) diamond patterns fall within the crossing-free resolution fragment. We can adapt such proofs without increasing their size as showcased in Figure 6.21. This particular adaptation is entailed by the fact that each diamond is a crossing-free derivation and more specifically an ensuing derivation of a non read-once clause.

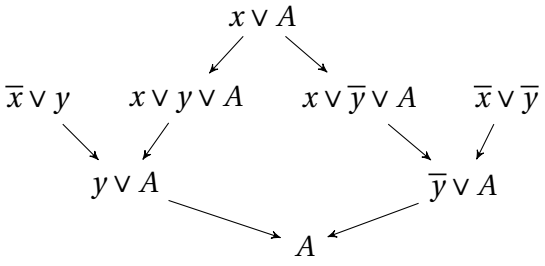


Figure 6.20: Adaptation of a diamond pattern as a regular tree resolution proof using the split rule

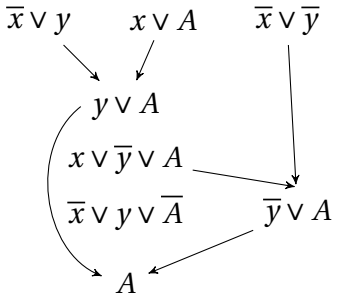


Figure 6.21: Adaptation of a diamond pattern as a crossing-free resolution proof

6.5 Certificates & Explanations for Max-SAT

In this section, we briefly describe other results which are detailed in [Py21]. We first present some results on certificate generation for Max-SAT. Indeed, our adaptations in Section 6.2 led to the introduction of the first independent proof builder for Max-SAT, called MS-Builder, which relies on successive calls to a SAT solver. Our work also extends to studying the inferential power of proof system for Max-SAT through the notion of explainability.

6.5.1 Certificates for Max-SAT

Our adaptations in Section 6.2 led to the introduction of the first independent proof builder for Max-SAT. This builder, called MS-Builder, generates certificates for the Max-SAT Problem in the particular form of a Max-SAT-equivalence-preserving transformation from the initial formula into a formula composed of a set of empty clauses and a satisfiable sub-formula, plus a model for this satisfiable sub-formula. Given an initial formula, the idea behind MS-Builder is to iteratively call a SAT oracle to get a resolution refutation which is first adapted into a max-refutation and then applied to the current formula. The builder halts when the SAT oracle returns that the current formula is satisfiable, in which case a model can be provided. The complete sequence of transformations which generates k empty clauses is a proof that the Max-SAT optimum is at least k while the the model ensures that it is possible to falsify exactly k clauses and, therefore, that the Max-SAT optimum k.

The proof builder can be also augmented with an additional preprocessing step before performing an adaptation. As resolution refutations are computed using SAT oracles, we can to fix non read-once unit clauses in the returned refutations when possible. Indeed, SAT algorithms rely heavily on UP [8] which simply consists in propagating literals in unit clauses in the whole formula, because satisfying those literal is necessary to satisfy the formula. Applying unit propagation can be seen as the use of a particular unit clause in several resolution steps. Transforming resolution refutations to fix non-read-once unit clauses can therefore be a useful preprossessing technique. The soundness of such a mechanism is established in the following proposition. We also provide two examples illustrating

[8] refer to Section 2.2.1

respectively the preprocessing mechanism and the generation of certificates by the proof builder.

Proposition 6.4. *Let ϕ be an unsatisfiable CNF formula, π be a resolution refutation of ϕ and C be a non-read-once unit clause in π s.t. every path in π from C to $\square$ contains no irregularity on $var(C)$. There exists a resolution refutation of ϕ such that C is read-once containing $O(|\pi|)$ inference steps.*

Proof. We assume w.l.o.g. that $C = x$. Since every path in π from C to $\square$ contains no irregularity on $var(C)$, we can delete the resolution steps on clause C and prune the refutation accordingly to deduce the clause $C' = \overline{x}$ instead of $\square$. We can now add a resolution step on clauses C and C' to get a resolution refutation of ϕ containing $O(|P|)$ inference steps.

■

Example 6.10. *We consider the $\phi = (x_1) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$ and the resolution refutation of π represented in Figure 2.7. Note how the non-read-once unit clause $C = x_1$ can be fixed by the preprocessing described in Proposition 6.4. To this end, we simply need to delete the unit clause and prune the proof and then re-inject it at the end of the proof to deduce the empty clause. The resulting refutation after preprocessing is represented on the left in Figure 6.22. Since it is a read-once proof, it can be trivially adapted into the max-refutation represented on the right in the same figure.*

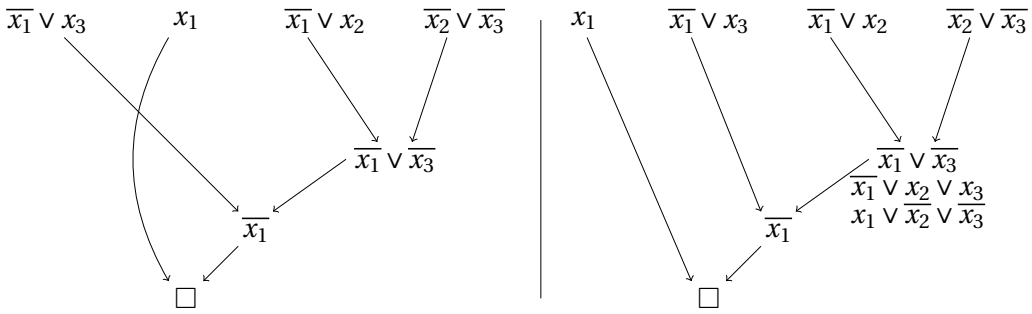


Figure 6.22: Preprocessing proofs in MS-builder

Example 6.11. *We consider the formula $\phi = (x_1) \wedge (\overline{x_1} \vee x_3) \wedge (\overline{x_1} \vee x_2)(\overline{x_2}) \wedge (\overline{x_3}) \wedge (x_2 \vee x_3)$ passed to MS-Builder in standard WCNF format [9]. MS-Builder calls a SAT oracle for the first time which returns that the initial formula is unsatisfiable alongside a read-once refutation π represented on the left in Figure 6.23. π is then adapted into the max-refutation represented on the right in the same figure. After applying the max-refutation, we obtain the current formula $\phi = (\overline{x_1} \vee x_3) \wedge (x_1) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3}) \wedge \square$. MS-Builder calls the SAT oracle for the second time without including the empty clause. The formula is unsatifiable and the oracle returns the resolution refutation represented in Figure 2.7 which is preprocessed to fix the non-read-once unit clause and adapted as specified in Example 6.10. After applying*

[9] https://maxsat-evaluations.github.io/2022/rules.html

the max-refutation, the current formula becomes $\phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge \square \wedge \square$. *MS-Builder calls a SAT oracle for the third and last time since it returns that the current formula without the empty clauses is satisfied by the assignment* $\alpha = \{\overline{x_1}, \overline{x_2}, \overline{x_3}\}$.

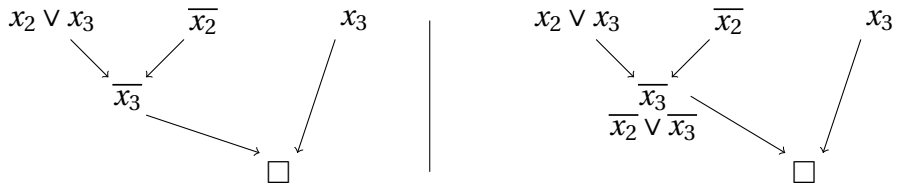


Figure 6.23: Adaptation of a resolution refutation in MS-Builder

The proof builder is also coupled with an extendable proof checker, called MS-checker, which takes the CNF formula and a Max-SAT certificate as input. After reading the formula, it verifies that the proposed inference rules are correct and that the premises are still in the formula then applies the transformation. Finally, it checks if the truth assignment satisfies the final formula without considering the empty clauses. Both the builder and the checker were implemented in C++ [10] while the resolution refutations were computed using Booleforce [11]. An experimental evaluation was conducted on the benchmarks of the unweighted and weighted complete tracks of the 2020 Max-SAT Evaluation [12]. The unweighted (resp. weighted) benchmark consists of 576 (resp. 600) industrial and crafted instances. With a timeout set to 1 hour, MS-Builder has succeeded to construct full proofs for 163 unweighted (resp. 144 for weighted) instances over 576 (resp. 600). More interestingly, MS-Builder has succeeded to build at least half of the proofs, with respect to the optimum of the input formula, for 302 unweighted instances over 463 (resp. 326 weighted instances over 489) for which the optimum cost is known. We refer the reader to [Py21] for the detailed experimental evaluation.

6.5.2 Explanations for Max-SAT

Our work also extends to the study of the inferential power of Max-SAT proof systems. Hereafter, we succinctly describe the major results and the reader can refer to [Py21] for the detailed results and proofs. First, we introduce the notion of explainable clauses which intuitively represent information that can be deduced from a formula. If we exhibit an equivalence-preserving transformation π from ϕ to $C \wedge \phi'$, we say that π is an explanation of C in ϕ. Such proof is sufficient to certify that the wanted information, i.e., a clause, can be inferred soundly from the formula. We characterize explainable clauses in Theorem 6.4. Inferential completeness is strongly related to the notion of explanation. Naturally, a proof system is inferentially complete if it is possible to provide an explanation using its rules for every explainable clause. Recall that, as stated in Proposition 3.3, Max-SAT resolution is not inferentially complete. We will therefore use a new proof system for explanation, called

[10] Source code available on https://pageperso.lis-lab.fr/matthieu.py/en/software.html
[11] http://fmv.jku.at/booleforce/index.html
[12] https://maxsat-evaluations.github.io/2020/

the Explanation Calculus (ExC), composed of two rules: the symmetric cut rule [13] and the expansion rule [14]. The expansion rule is defined below and proved sound for Max-SAT and it can be intuitively considered as an extension of the weakening rule used in the context of SAT [15].

Definition 6.7 (Explainable Clause)**.** *Let ϕ be a CNF formula and C be a non-tautological clause, we say that C is explainable in ϕ if there exists a CNF formula ϕ' such that $\phi \equiv C \wedge \phi'$ otherwise we say that C is unexplainable in ϕ.*

Theorem 6.4. *Let ϕ be a CNF formula and C be a non-tautological clause. C is explainable in ϕ if and only if $\exists C' \in \phi$ such that C' doesn't oppose C and $\forall x \notin var(C)$, $C \vee x$ and $C \vee \overline{x}$ are both explainable in ϕ.*

Definition 6.8 (Expansion)**.** *Given a clause C and a set of literals $B = \{b_1, \ldots, b_k\}$, the expansion rule applied on C w.r.t B is defined as follows:*

$$\frac{C}{\begin{array}{c} CC_1 = A \vee \overline{b_1} \\ CC_2 = A \vee b_1 \vee \overline{b_2} \\ \ldots \\ CC_k = A \vee b_1 \vee \ldots \vee b_{k-1} \vee \overline{b_k} \\ Ce = A \vee B \end{array}}$$

where Ce is the expanded clause and $CC_1, \ldots, CC_k$ are compensation clauses.

Proposition 6.5. *The expansion rule is sound for Max-SAT.*

Definition 6.9 (Explanation Calculus)**.** *The Explanation Calculus (ExC) is composed of two rules: symmetric cut and expansion.*

Proposition 6.6. *ExC is inferentially complete.*

We study the relation between **ExC** and other proof systems. It is important to note that we are interested in the notion of inference in the context of Max-SAT on a wider scope, i.e., we want to study the general inferential power of proof systems and compare them in this sense and not just focus on their refutational power. We prove that ExC i-p-simulates [16] symmetric cut + split, Max-SAT resolution and ResS [LR20b]. Finally, we observe that it is possible to weaken Max-SAT resolution into the symmetric-cut without loosing any power if we add expansion. These results are summarized in Figure 6.24.

Next, we show that, given a CNF formula ϕ, we can exhibit an explanation containing $O(2^{|Var(\phi)|})$ inference steps in ExC for any explainable clause C in ϕ. This result is established in Theorem 6.5 by showing that it is possible to reverse infer the wanted transformation starting from C instead of ϕ as specified in the characterization proved in Theorem 6.4.

[13] refer to Definition 3.12
[14] Note that the expansion rule used here is different from the expansion rewriting used in Section 6.3.2.
[15] refer to Definition 2.22
[16] refer to Definition 3.24

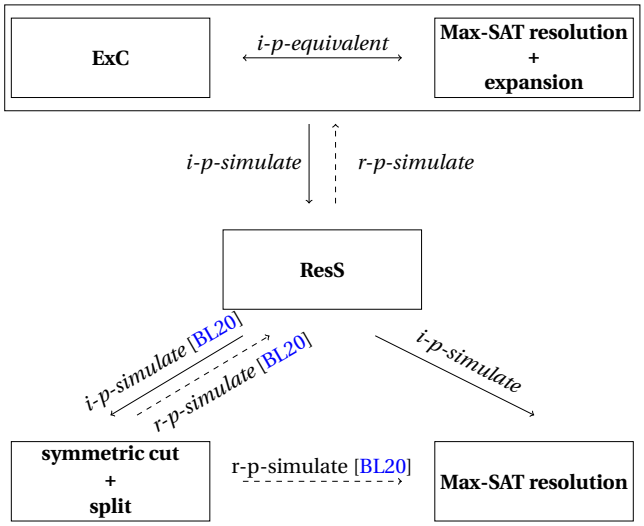


Figure 6.24: Relationship between **ExC** and other proof systems

This result provides a better bound on the size of the proofs than the existing result saturation [17]. We illustrate how to generate clause explanations in ExC in Example 6.12. Note that the notion of explainability as well as our results can be extended to formulas as detailed in [PCH21d; Py21].

Theorem 6.5. *Let ϕ be a CNF formula and C be an explainable clause in ϕ. There exists an explanation of C in ϕ using ExC rules containing $O(2^{|var(\phi)|})$ inference steps.*

Example 6.12. *We consider the CNF formula $\phi = (x_1 \vee x_2) \wedge (x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3) \wedge (\overline{x_1})$ and we want to explain $C = x_1$. As showcased in the proof of Theorem 6.5, we can construct the explanation of C in ExC represented in Figure 6.25. Indeed, considering the variables in lexicographic order, since C is not subsumed by a clause in ϕ nor it is opposed to all the clauses, we thus try to explain $x_1 \vee x_2$ and $x_1 \vee \overline{x_2}$ in ϕ. While the former is trivially explained since it is in the formula, the latter not subsumed by a clause nor opposed to all the clauses. Therefore, we need to explain the clauses $x_1 \vee \overline{x_2} \vee x_3$ and $x_1 \vee \overline{x_2} \vee \overline{x_3}$. The first clause is subsumed by x_3 and therefore it can be explained by expansion while the second one is in the formula and therefore is trivially explained. We conclude that $\phi \vdash (\overline{x_1} \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_1) \wedge (\overline{x_1})$ by applying the transformation represented in Figure 6.25. Notice how the explanation is constructed from the bottom to the top starting from C. Furthermore, clauses $(\overline{x_1} \vee x_3)$ and $(x_1 \vee x_2 \vee x_3)$ play the role of compensation clauses, essential to preserve Max-SAT equivalence.*

Our result on clause explainability can also be used to build certificates for Max-SAT as established in Theorem 6.6. Intuitively, the idea is to iteratively try to explain the empty clause until it is no longer explainable in the current formula. By doing this, we iteratively infer as many empty clauses as the optimum of the initial formula. Finally, we mention that our results can be easily extended to weighted Max-SAT as detailed in [PCH21d; Py21].

[17] refer to Section 3.3.2

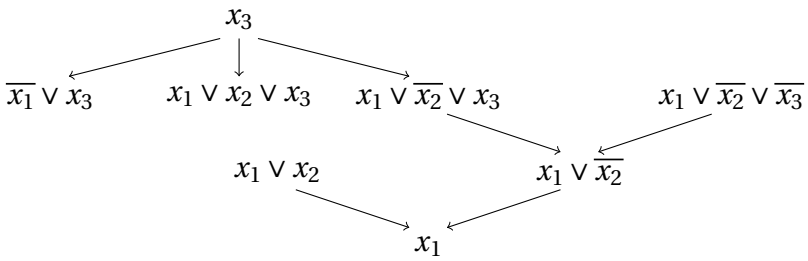


Figure 6.25: Explanation of clause $C = x_1$ in ϕ

Theorem 6.6. *Let ϕ be a CNF formula. we can deduce $\phi \vdash_{ExC} \underbrace{\square \wedge \cdots \wedge \square}_{opt(\phi)} \wedge \phi'$ where ϕ' is satisfiable in $O(|\phi| \times 2^{|var(\phi)|})$ inference steps.*

6.6 Conclusion & Future Work

In this chapter, we tackled the open problem of adapting resolution refutations into proofs valid for Max-SAT. In particular, we proposed linear adaptations with respect to the size of the resolution refutations in the following cases: regular tree resolution, tree resolution and semi-tree resolution. These results are achieved by augmenting Max-SAT resolution with the split rule which enabled us to duplicate clauses by adding literals when necessary. We have also generalized our adaptation to unrestricted resolution refutations, even though the proposed transformation can produce a max-refutation whose size is exponential in the worst case. We also proposed a new class of resolution, called crossing-free resolution, for which an adaptation to Max-SAT resolution proofs is possible without a substantial increase in the size of the proofs. Furthermore, we relied on particular resolution proofs, called (k-stacked) diamond patterns to study the features and limits of our proposed adaptations specifically in the unrestricted case. Note that our results on proof adaptations also include the attempt in [PCH21b; PCH22c] where we study adaptations from SAT proofs to Max-SAT proofs through the lens of SAT oracles, exploiting their ability to return resolution refutations, although we did not manage to provide formal guarantees on the resulting proofs in this case. Our results contribute to the difficult open problem of adapting resolution proofs to Max-SAT resolution proofs without increasing their size thus helping to bridge the gap between resolution for SAT and Max-SAT. Furthermore, unlike SAT solvers, Max-SAT solvers are still not able to output certificates in the form of Max-SAT equivalent proofs mainly due to the variety of solving paradigms and due to the theoretical gap between SAT and Max-SAT resolution. Our work has been useful in this regard as it led to the introduction of the first independent Max-SAT proof builder, able to compute certificates by relying on successive calls to a SAT oracle [PCH21a; Py21]. Our work also extends beyond the refutational power of proof systems for Max-SAT as our results on explainability provide further insights on the inferential power of proof systems for Max-SAT [PCH21d; Py21].

As future work, it would be interesting to characterize a larger intersection between SAT and Max-SAT resolution by proving that an adaptation of an extended refinement of

resolution (ideally unrestricted resolution) is possible without a substantial increase in the size of the proofs. We could look into whether this can be achieved by augmenting Max-SAT resolution with other inference rules like the split rule. Overall, the existence of an adaptation that does not increase substantially the size of an unrestricted resolution proof remains an open question and it would be relevant to further investigate this topic either by exhibiting a polynomial adaptation or refuting its existence. Other prospects of our work include using the proposed adaptations in the context SAT-based algorithms for Max-SAT. For instance, it would be interesting to extend core-guided algorithms with our linear adaptations when deemed possible in the spirit of [HM11]. Our proof builder could also benefit from more advanced processing techniques used in SAT-based solvers such as stratification [18] or from new techniques specifically devised for the builder such as extending the preprocessing mechanism in Proposition 6.4 to non-unit clauses.

[18] refer to Section 3.2.2.2

Conclusion

Our work in this manuscript revolved around two major logic-based formalisms: the well-known SAT problem and its natural optimization extension Max-SAT. Our contributions aimed to deepen our understanding of the power and limits of reasoning and inference and how to efficiently apply them in the context of the SAT and Max-SAT paradigms. Our first contribution involved the evaluation of different strategies which take advantage of the restart mechanism to combine two state-of-the-art heuristics, i.e., VSIDS and CHB. In particular, we proposed a reinforcement learning approach relying on a Multi-Armed Bandit (MAB) framework for adaptive branching in SAT. We used a reward function which evaluates the capacity of the heuristics to reach conflicts quickly and efficiently. Our results indicate that Upper Confidence Bound (UCB) strategies such as UCB1 and MOSS which conduct stochastic inference on the mean reward of each heuristic in the context of our MAB framework are able to achieve a considerable gain in terms of solved instances and solving time. Our work in this regard touches upon the major challenge of combining or finding a middle ground between symbolic and numeric AI, which are often seen as incompatible or opposed with one another. Recent works including ours in the context of SAT [CHT21a; CHT21b; CHT21d; CHT22a] and CSP [CHT20; CHT21c] suggest that there is an interest in applying techniques originating from non-symbolic AI and particularly reinforcement learning in combinatorial problem solving. It would be therefore interesting to further investigate this line of research by trying to apply similar frameworks to enhance the reasoning power of SAT solvers. One can target other components in modern solvers such as restarts, backtracking and clause-database management among others. It would be also relevant to investigate whether such frameworks can be incorporated both in BnB solvers and SAT based solvers for Max-SAT.

Our second contribution consisted in thoroughly investigating the power of inference in the context of Max-SAT BnB through the property of UP-resilience. We showed that this notion can help quantify the impact of Max-SAT resolution transformations of ISs on the SUP mechanism. Our observation on the direct impact of the order of application of Max-SAT resolution during the ISs transformation on its UP-resilience, led to a more in-depth understanding of the efficiency (or the lack thereof) of learning mechanisms used during LB estimation in the literature including the traditional patterns [LMP07; Li+10a] and the recently introduced UCS patterns [AH14d]. In particular, we showed that, while the traditional three patterns are UP-resilient with respect to any order of application of Max-SAT resolution, the recently introduced UCS patterns are not necessarily UP-resilient. They are nevertheless UP resilient with respect to other newly introduced orders. Our results thus contribute to explain the limits of the current learning mechanisms and to shed the light on the recent empirical observations in [AH14d]. To our best knowledge, this is the first work in which UP-resilience is used to theoretically characterize the transformations by Max-SAT

resolution with respect to the possible orders of its application. Indeed, this can provide new insights on how to extend learning mechanisms used in Max-SAT BnB solvers. It would be relevant to study whether Max-SAT inference under the Max-SAT resolution rule can be efficiently incorporated in the recent BnB paradigm introduced in [Li+21a; Li+22a] either to further guide the exploration of the search or simply to preprocess the formula before the actual search. It would also be relevant to study whether similar learning mechanisms can be used in the context of SAT-based solvers and specifically those which already exploit Max-SAT resolution to transform cores returned by SAT solvers [HM11; NB14].

Our contributions also extend to the study of the refutational and inferential power of proof systems for Max-SAT and their relation with SAT inference. We particularly tackled an open problem since the introduction of Max-SAT resolution as a complete calculus for Max-SAT in [BLM06; BLM07] which consists in adapting resolution refutations into max-refutations of reasonable size. We proposed linear adaptations with respect to the size of the resolution refutations in the case of regular tree resolution, tree resolution and semi-tree resolution when Max-SAT resolution is augmented with the split rule. Moreover, we showed that an adaptation of general resolution refutations is possible but with a worst case exponential blow-up in the size of the proofs. We also proposed a new class of resolution, called crossing-free resolution, for which an adaptation using exclusively Max-SAT resolution is possible without a substantial increase in the size of the proofs. These results also helped to devise an independent proof builder for Max-SAT, called MS-Builder, which is able to compute certificates for Max-SAT by relying on successive calls to a SAT oracle. Our work also extends beyond the refutational power of proof systems for Max-SAT as we study the inferential power of Max-SAT proof systems through the notion of explainability. As future work, it would be interesting to characterize a larger intersection between SAT and Max-SAT resolution by proving that an adaptation of an extended refinement of resolution (ideally unrestricted resolution) is possible without a substantial increase in the size of the proofs. We could look into whether this can be achieved by augmenting Max-SAT resolution with other inference rules like the split rule. Overall, the existence of an adaptation that does not increase substantially the size of an unrestricted resolution proof remains an open question and it would be relevant to further investigate this topic either by exhibiting a polynomial adaptation or refuting its existence. Other prospects of our work include using the proposed adaptations in the context of SAT-based algorithms for Max-SAT. As mentioned above, increasing the knowledge on Max-SAT inference and particularly Max-SAT resolution can be used to extend and enhance the efficiency of SAT-based algorithms for Max-SAT. For instance, it would be interesting to extend core-guided algorithms with our linear adaptations when deemed efficient compared to cardinality encodings or simply in the spirit of [HM11]. The proof builder could also benefit from more advanced mechanisms based on or inspired from SAT-based solvers for Max-SAT [BJM21]. Finally, it would be relevant to extend some of our results on close problems such as Minimum Satisfiability (Min-SAT) [LM22] or Constraint Optimization (COP) [RBW06].

Bibliography

[Abr15] André Abramé. “Max-résolution et apprentissage pour la résolution du problème de satisfiabilité maximum”. fr. PhD thesis. Aix-Marseille University, 2015 (cit. on pp. 19, 86, 96, 99, 114, 138, 149).

[AH14a] André Abramé and Djamal Habet. “Efficient Application of Max-SAT Resolution on Inconsistent Subsets”. In: *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*. Ed. by Barry O’Sullivan. Vol. 8656. Lecture Notes in Computer Science. Springer, 2014, pp. 92–107. DOI: 10.1007/978-3-319-10428-7_10 (cit. on pp. 99, 137).

[AH14b] André Abramé and Djamal Habet. “Local Max-Resolution in Branch and Bound Solvers for Max-SAT”. In: *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*. IEEE Computer Society, 2014, pp. 336–343. DOI: 10.1109/ICTAI.2014.58 (cit. on pp. 19, 93, 114).

[AH14c] André Abramé and Djamal Habet. “Maintaining and Handling All Unit Propagation Reasons in Exact Max-SAT Solvers”. In: *Proceedings of the Seventh Annual Symposium on Combinatorial Search, SOCS 2014, Prague, Czech Republic, 15-17 August 2014*. Ed. by Stefan Edelkamp and Roman Barták. AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/SOCS/SOCS14/paper/view/8908 (cit. on p. 92).

[AH14d] André Abramé and Djamal Habet. “On the Extension of Learning for Max-SAT”. In: *STAIRS 2014 - Proceedings of the 7th European Starting AI Researcher Symposium, Prague, Czech Republic, August 18-22, 2014*. Ed. by Ulle Endriss and João Leite. Vol. 264. Frontiers in Artificial Intelligence and Applications. IOS Press, 2014, pp. 1–10. DOI: 10.3233/978-1-61499-421-3-1 (cit. on pp. 19, 95, 96, 99, 135, 138, 140, 144, 149, 180).

[AH15a] André Abramé and Djamal Habet. “ahmaxsat: Description and Evaluation of a Branch and Bound Max-SAT Solver”. en. In: *Journal on Satisfiability, Boolean Modeling and Computation* 9 (2015), pp. 89–128 (cit. on pp. 86, 99, 114).

[AH15b] André Abramé and Djamal Habet. “On the Resiliency of Unit Propagation to Max-Resolution”. In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Ed. by Qiang Yang and Michael J. Wooldridge. AAAI Press, 2015, pp. 268–274. URL: http://ijcai.org/Abstract/15/044 (cit. on pp. 19, 96, 98, 99, 114, 135–138).

[AH16] André Abramé and Djamal Habet. “Learning Nobetter Clauses in Max-SAT Branch and Bound Solvers”. In: *28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016, San Jose, CA, USA, November 6-8, 2016*. IEEE Computer Society, 2016, pp. 452–459. DOI: 10.1109/ICTAI.2016.0075 (cit. on p. 91).

[AHT17] André Abramé, Djamal Habet, and Donia Toumi. “Improving configuration checking for satisfiable random k-SAT instances”. In: *Ann. Math. Artif. Intell.* 79.1-3 (2017), pp. 5–24. DOI: 10.1007/s10472-016-9515-9 (cit. on pp. 62–65).

[AN14] Roberto Javier Asín Achá and Robert Nieuwenhuis. “Curriculum-based course timetabling with SAT and MaxSAT”. In: *Ann. Oper. Res.* 218.1 (2014), pp. 71–91. DOI: 10.1007/s10479-012-1081-x (cit. on p. 85).

[AM02] Dimitris Achlioptas and Cristopher Moore. “The Asymptotic Order of the Random k -SAT Threshold”. In: *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*. IEEE Computer Society, 2002, pp. 779–788. DOI: 10.1109/SFCS.2002.1182003 (cit. on p. 66).

[Ach09] Tobias Achterberg. “SCIP: solving constraint integer programs”. In: *Math. Program. Comput.* 1.1 (2009), pp. 1–41. DOI: 10.1007/s12532-008-0001-1 (cit. on p. 110).

[AKS04] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. “PRIMES is in P”. In: *Annals of Mathematics* 160.2 (Sept. 2004), pp. 781–793. DOI: 10.4007/annals.2004.160.781 (cit. on p. 31).

[Agr95] Rajeev Agrawal. “Sample Mean Based Index Policies with O(log n) Regret for the Multi-Armed Bandit Problem”. In: *Advances in Applied Probability* 27.4 (1995), pp. 1054–1078. ISSN: 00018678. URL: http://www.jstor.org/stable/1427934 (cit. on p. 79).

[AG17] Shipra Agrawal and Navin Goyal. “Near-Optimal Regret Bounds for Thompson Sampling”. In: *J. ACM* 64.5 (2017). ISSN: 0004-5411. DOI: 10.1145/3088510 (cit. on p. 77).

[Aho+74] A.V. Aho, A.V.A. AHO, J.E. Hopcroft, et al. *The Design and Analysis of Computer Algorithms*. Addison-Wesley series in computer science and information processing. Addison-Wesley Publishing Company, 1974. ISBN: 9780201000290. URL: https://books.google.fr/books?id=SJJQAAAAMAAJ (cit. on p. 27).

[Ale+02] Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, et al. “An Exponential Separation between Regular and General Resolution”. In: *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*. STOC ’02. Montreal, Quebec, Canada: Association for Computing Machinery, 2002, pp. 448–456. ISBN: 1581134959. DOI: 10.1145/509907.509974 (cit. on p. 73).

[AM21] Haifa Hamad Alkasem and Mohamed El Bachir Menai. “A Stochastic Local Search Algorithm for the Partial Max-SAT Problem Based on Adaptive Tuning and Variable Depth Neighborhood Search”. In: *IEEE Access* 9 (2021), pp. 49806–49843. DOI: 10.1109/ACCESS.2021.3068824 (cit. on p. 111).

[Alo+02] Fadi A. Aloul, Arathi Ramani, Igor L. Markov, et al. “Generic ILP versus specialized 0-1 ILP: an update”. In: *Proceedings of the 2002 IEEE/ACM International Conference on Computer-aided Design, ICCAD 2002, San Jose, California, USA, November 10-14, 2002*. Ed. by Lawrence T. Pileggi and Andreas Kuehlmann. ACM / IEEE Computer Society, 2002, pp. 450–457. DOI: 10.1145/774572.774638 (cit. on p. 101).

[AMP03] Teresa Alsinet, Felip Manya, and Jordi Planes. “Improved branch and bound algorithms for Max-SAT”. In: *Proc. of 6th International Conference on the Theory and Applications of Satisfiability Testing (SAT)*. 2003, pp. 408–415 (cit. on p. 87).

[AMP04] Teresa Alsinet, Felip Manyà, and Jordi Planes. “A Max-SAT Solver with Lazy Data Structures”. In: *Advances in Artificial Intelligence - IBERAMIA 2004, 9th Ibero-American Conference on AI, Puebla, Mexico, November 22-26, 2004, Proceedings*. Ed. by Christian Lemaître, Carlos A. Reyes García, and Jesús A. González. Vol. 3315. Lecture Notes in Computer Science. Springer, 2004, pp. 334–342. DOI: 10.1007/978-3-540-30498-2_34 (cit. on p. 88).

[AMP05] Teresa Alsinet, Felip Manyà, and Jordi Planes. “Improved Exact Solvers for Weighted Max-SAT”. In: *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*. Ed. by Fahiem Bacchus and Toby Walsh. Vol. 3569. Lecture Notes in Computer Science. Springer, 2005, pp. 371–377. DOI: 10.1007/11499107_27 (cit. on pp. 87, 88).

[Alt+21] Fabrizio Altarelli, Rémi Monasson, Guilhem Semerjian, et al. “Connections to Statistical Physics”. In: *Handbook of Satisfiability - Second Edition*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, et al. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 859–901. DOI: 10.3233/FAIA201006 (cit. on p. 39).

[ADR15] Mario Alviano, Carmine Dodaro, and Francesco Ricca. “A MaxSAT Algorithm Using Cardinality Constraints of Bounded Size”. In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Ed. by Qiang Yang and Michael J. Wooldridge. AAAI Press, 2015, pp. 2677–2683. URL: http://ijcai.org/Abstract/15/379 (cit. on p. 105).

[AMM22a] Tasniem Nasser Alyahya, Mohamed El Bachir Menai, and Hassan Mathkour. “On the Structure of the Boolean Satisfiability Problem: A Survey”. In: *ACM Comput. Surv.* 55.3 (2022). ISSN: 0360-0300. DOI: 10.1145/3491210 (cit. on p. 66).

[And+12] Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, et al. “Unsatisfiability-based optimization in clasp”. In: *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary*. Ed. by Agostino Dovier and Vítor Santos Costa. Vol. 17. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012, pp. 211–221. DOI: 10.4230/LIPIcs.ICLP.2012.211 (cit. on p. 106).

[Ans+13] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, et al. “Improving WPM2 for (Weighted) Partial MaxSAT”. In: *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*. Ed. by Christian Schulte. Vol. 8124. Lecture Notes in Computer Science. Springer, 2013, pp. 117–132. DOI: 10.1007/978-3-642-40627-0_12 (cit. on pp. 108, 109).

[Ans+14] Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, et al. “The Fractal Dimension of SAT Formulas”. In: *Automated Reasoning*. Ed. by Stéphane Demri, Deepak Kapur, and Christoph Weidenbach. Cham: Springer International Publishing, 2014, pp. 107–121. ISBN: 978-3-319-08587-6 (cit. on p. 69).

[Ans+19] Carlos Ansótegui, Maria Luisa Bonet, Jesús Giráldez-Cru, et al. “Community Structure in Industrial SAT Instances”. In: *J. Artif. Intell. Res.* 66 (2019), pp. 443–472. DOI: 10.1613/jair.1.11741 (cit. on p. 68).

[ABL09a] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. “On Solving MaxSAT Through SAT”. In: *Artificial Intelligence Research and Development, Proceedings of the 12th International Conference of the Catalan Association for Artificial Intelligence, CCIA 2009, October 21-23, 2009, Vilar Rural de Cardona (El Bages), Cardona, Spain*. Ed. by Sandra A. Sandri, Miquel Sànchez-Marrè, and Ulises Cortés. Vol. 202. Frontiers in Artificial Intelligence and Applications. IOS Press, 2009, pp. 284–292. DOI: 10.3233/978-1-60750-061-2-284 (cit. on p. 105).

[ABL09b] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. “Solving (Weighted) Partial MaxSAT through Satisfiability Testing”. In: *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*. Ed. by Oliver Kullmann. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009, pp. 427–440. DOI: 10.1007/978-3-642-02777-2_39 (cit. on p. 105).

[ABL09c] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. “Towards Industrial-like Random SAT Instances”. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. IJCAI’09. Pasadena, California, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 387–392 (cit. on pp. 67, 68).

[ABL10a] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. “A New Algorithm for Weighted Partial MaxSAT”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. Ed. by Maria Fox and David Poole. AAAI Press, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1774 (cit. on p. 104).

[ABL10b] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. “A New Algorithm for Weighted Partial MaxSAT”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. Ed. by Maria Fox and David Poole. AAAI Press, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1774 (cit. on p. 105).

[ABL13] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. “SAT-based MaxSAT algorithms”. In: *Artificial Intelligence* 196 (2013), pp. 77–105. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2013.01.002 (cit. on pp. 84, 100, 104).

[ABL19] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. “Phase Transition in Realistic Random SAT Models 1”. In: *Artificial Intelligence Research and Development*. IOS Press, 2019, pp. 213–222 (cit. on p. 67).

[ABL22] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. “Scale-Free Random SAT Instances”. In: *Algorithms* 15.6 (2022), p. 219. DOI: 10.3390/a15060219 (cit. on p. 68).

[ABL09d] Carlos Ansótegui, María Luisa Bonet, and Jordi Levy. “On the Structure of Industrial SAT Instances”. In: *Principles and Practice of Constraint Programming - CP 2009*. Ed. by Ian P. Gent. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 127–141 (cit. on p. 68).

[ADG15] Carlos Ansótegui, Frédéric Didier, and Joel Gabàs. “Exploiting the Structure of Unsatisfiable Cores in MaxSAT”. In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Ed. by Qiang Yang and Michael J. Wooldridge. AAAI Press, 2015, pp. 283–289. URL: http://ijcai.org/Abstract/15/046 (cit. on p. 105).

[AG13] Carlos Ansótegui and Joel Gabàs. “Solving (Weighted) Partial MaxSAT with ILP”. In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings*. Ed. by Carla P. Gomes and Meinolf Sellmann. Vol. 7874. Lecture Notes in Computer Science. Springer, 2013, pp. 403–409. URL: http://www.cis.cornell.edu/ics/cpaior2013/pdfs/ansotegui.pdf (cit. on p. 111).

[Ans+16] Carlos Ansótegui, Joel Gabàs, Yuri Malitsky, et al. “MaxSAT by improved instance-specific algorithm configuration”. In: *Artif. Intell.* 235 (2016), pp. 26–39. DOI: 10.1016/j.artint.2015.12.006 (cit. on p. 111).

[AGL12] Carlos Ansótegui, Jesús Giráldez-Cru, and Jordi Levy. “The Community Structure of SAT Formulas”. In: *Theory and Applications of Satisfiability Testing – SAT 2012*. Ed. by Alessandro Cimatti and Roberto Sebastiani. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 410–423 (cit. on p. 68).

[Ans+15a] Carlos Ansótegui, Jesús Giráldez-Cru, Jordi Levy, et al. “Using Community Structure to Detect Relevant Learnt Clauses”. In: *Theory and Applications of Satisfiability Testing – SAT 2015*. Ed. by Marijn Heule and Sean Weaver. Cham: Springer International Publishing, 2015, pp. 238–254. ISBN: 978-3-319-24318-4 (cit. on p. 56).

[Ans+15b] Carlos Ansótegui, Jesús Giráldez-Cru, Jordi Levy, et al. “Using Community Structure to Detect Relevant Learnt Clauses”. In: *Theory and Applications of Satisfiability Testing – SAT 2015*. Ed. by Marijn Heule and Sean Weaver. Cham: Springer International Publishing, 2015, pp. 238–254. ISBN: 978-3-319-24318-4 (cit. on p. 68).

[AL11] Carlos Ansótegui and Jordi Levy. “On the Modularity of Industrial SAT Instances”. In: *Artificial Intelligence Research and Development - Proceedings of the 14th International Conference of the Catalan Association for Artificial Intelligence, Lleida, Catalonia, Spain, October 26-28, 2011*. Ed. by Cèsar Fernández, Hector Geffner, and Felip Manyà. Vol. 232. Frontiers in Artificial Intelligence and Applications. IOS Press, 2011, pp. 11–20. DOI: 10.3233/978-1-60750-842-7-11 (cit. on p. 68).

[AL21] Carlos Ansótegui and Jordi Levy. “Reducing SAT to Max2SAT”. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*. Ed. by Zhi-Hua Zhou. ijcai.org, 2021, pp. 1367–1373. DOI: 10.24963/ijcai.2021/189 (cit. on p. 59).

[AMS14] Carlos Ansótegui, Yuri Malitsky, and Meinolf Sellmann. “MaxSAT by Improved Instance-Specific Algorithm Configuration”. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*. Ed. by Carla E. Brodley and Peter Stone. AAAI Press, 2014, pp. 2594–2600. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8621 (cit. on p. 111).

[AH11] Alejandro Arbelaez and Youssef Hamadi. “Improving Parallel Local Search for SAT”. In: *Learning and Intelligent Optimization*. Ed. by Carlos A. Coello Coello. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 46–60. ISBN: 978-3-642-25566-3 (cit. on p. 66).

[Arg+08] Josep Argelich, Alba Cabiscol, Inês Lynce, et al. “Encoding Max-CSP into Partial Max-SAT”. In: *38th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2008), 22-23 May 2008, Dallas, Texas, USA*. IEEE Computer Society, 2008, pp. 106–111. DOI: 10.1109/ISMVL.2008.22 (cit. on p. 85).

[Arg+09a] Josep Argelich, Alba Cabiscol, Inês Lynce, et al. “Regular Encodings from Max-CSP into Partial Max-SAT”. In: *ISMVL 2009, 39th International Symposium on Multiple-Valued Logic, 21-23 May 2009, Naha, Okinawaw, Japan*. IEEE Computer Society, 2009, pp. 196–202. DOI: 10.1109/ISMVL.2009.23 (cit. on p. 85).

[Arg+09b] Josep Argelich, Alba Cabiscol, Inês Lynce, et al. “Sequential Encodings from Max-CSP into Partial Max-SAT”. In: *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*. SAT ’09. Swansea, UK: Springer-Verlag, 2009, pp. 161–166. ISBN: 9783642027765. DOI: 10.1007/978-3-642-02777-2_17 (cit. on p. 85).

[AM06] Josep Argelich and Felip Manyà. “Exact Max-SAT solvers for over-constrained problems”. In: *J. Heuristics* 12.4-5 (2006), pp. 375–392. DOI: 10.1007/s10732-006-7234-9 (cit. on p. 88).

[AW02] Takao Asano and David P. Williamson. “Improved Approximation Algorithms for MAX SAT”. In: *J. Algorithms* 42.1 (2002), pp. 173–202. DOI: 10.1006/jagm.2001.1202 (cit. on p. 112).

[AF10] Dimitrios Athanasiou and Marco Alvarez Fernandez. *Recursive weight heuristic for random k-sat*. Tech. rep. Delft, Netherlands: Delft University of Technology, 2010 (cit. on p. 51).

[AL19] Albert Atserias and Massimo Lauria. “Circular (Yet Sound) Proofs”. In: *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*. Ed. by Mikolás Janota and Inês Lynce. Vol. 11628. Lecture Notes in Computer Science. Springer, 2019, pp. 1–18. DOI: 10.1007/978-3-030-24258-9_1 (cit. on pp. 71, 113).

[Aud+10] Gilles Audemard, Jean-Marie Lagniez, Bertrand Mazure, et al. “Boosting Local Search Thanks to cdcl”. In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by Christian G. Fermüller and Andrei Voronkov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 474–488. ISBN: 978-3-642-16242-8 (cit. on p. 66).

[AS09a] Gilles Audemard and Laurent Simon. “Predicting Learnt Clauses Quality in Modern SAT Solvers”. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. IJCAI’09. Pasadena, California, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 399–404 (cit. on pp. 42, 52).

[AS09b] Gilles Audemard and Laurent Simon. “Predicting Learnt Clauses Quality in Modern SAT Solvers”. In: *Proceedings of the 21st International Joint Conference on Artificial Intelligence*. IJCAI’09. Pasadena, California, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 399–404 (cit. on p. 56).

[AS12] Gilles Audemard and Laurent Simon. “Refining Restarts Strategies for SAT and UNSAT”. In: *Principles and Practice of Constraint Programming*. Ed. by Michela Milano. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 118–126. ISBN: 978-3-642-33558-7 (cit. on pp. 57, 119).

[AS14] Gilles Audemard and Laurent Simon. “Lazy Clause Exchange Policy for Parallel SAT Solvers”. In: *Theory and Applications of Satisfiability Testing – SAT 2014*. Ed. by Carsten Sinz and Uwe Egly. Cham: Springer International Publishing, 2014, pp. 197–205. ISBN: 978-3-319-09284-3 (cit. on p. 59).

[AS18] Gilles Audemard and Laurent Simon. “On the glucose SAT solver”. In: *International Journal on Artificial Intelligence Tools* 27.01 (2018), p. 1840001 (cit. on p. 56).

[AB09] Jean-Yves Audibert and Sébastien Bubeck. “Minimax Policies for Adversarial and Stochastic Bandits”. In: *Proceedings of the 22nd Annual Conference on Learning Theory (COLT)*. 2009. URL: https://www.microsoft.com/en-us/research/publication/minimax-policies-adversarial-stochastic-bandits/ (cit. on pp. 19, 77, 79).

[AMS09] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. “Exploration–exploitation tradeoff using variance estimates in multi-armed bandits”. In: *Theoretical Computer Science* 410.19 (2009). Algorithmic Learning Theory, pp. 1876–1902. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2009.01.016 (cit. on p. 79).

[Aue+95] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, et al. “Gambling in a rigged casino: The adversarial multi-armed bandit problem”. In: *Proceedings of IEEE 36th annual foundations of computer science*. IEEE. 1995, pp. 322–331 (cit. on p. 75).

[Aue+02] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, et al. “The nonstochastic multiarmed bandit problem”. In: *SIAM journal on computing* 32.1 (2002), pp. 48–77. DOI: https://doi.org/10.1137/S0097539701398375 (cit. on pp. 77, 132).

[ACF02] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. “Finite-time Analysis of the Multiarmed Bandit Problem”. In: *Mach. Learn.* 47.2-3 (2002), pp. 235–256. DOI: 10.1023/A:1013689704352 (cit. on pp. 19, 76, 77, 79).

[ABZ05] Adi Avidor, Ido Berkovitch, and Uri Zwick. “Improved Approximation Algorithms for MAX NAE-SAT and MAX SAT”. In: *Approximation and Online Algorithms, Third International Workshop, WAOA 2005, Palma de Mallorca, Spain, October 6-7, 2005, Revised Papers*. Ed. by Thomas Erlebach and Giuseppe Persiano. Vol. 3879. Lecture Notes in Computer Science. Springer, 2005, pp. 27–40. DOI: 10.1007/11671411_3 (cit. on p. 112).

[Bac+17] Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, et al. “Reduced Cost Fixing in MaxSAT”. In: *Principles and Practice of Constraint Programming*. Ed. by J. Christopher Beck. Cham: Springer International Publishing, 2017, pp. 641–651. ISBN: 978-3-319-66158-2 (cit. on p. 110).

[BJM21] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. “Maximum Satisfiabiliy”. In: *Handbook of Satisfiability - Second Edition*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, et al. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 929–991. DOI: 10.3233/FAIA201008 (cit. on pp. 19, 85, 100, 104, 181).

[Bac+15] I. Bachiri, J. Gaudreault, C. Quimper, et al. “RLBS: An Adaptive Backtracking Strategy Based on Reinforcement Learning for Combinatorial Optimization”. In: *Proceedings of ICTAI*. 2015, pp. 936–942 (cit. on p. 80).

[BB03] Olivier Bailleux and Yacine Boufkhad. “Efficient CNF Encoding of Boolean Cardinality Constraints”. In: *Principles and Practice of Constraint Programming – CP 2003*. Ed. by Francesca Rossi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 108–122. ISBN: 978-3-540-45193-8 (cit. on p. 106).

[BBR06] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. “A Translation of Pseudo Boolean Constraints to SAT”. In: *J. Satisf. Boolean Model. Comput.* 2.1-4 (2006), pp. 191–200. DOI: 10.3233/sat190021 (cit. on p. 100).

[BBP15] Amine Balafrej, Christian Bessiere, and Anastasia Paparrizou. “Multi-Armed Bandits for Adaptive Constraint Propagation”. In: *Proceedings of IJCAI*. 2015, pp. 290–296 (cit. on p. 80).

[BT83] Egon Balas and Paolo Toth. “Branch and bound methods for the traveling salesman problem”. In: (1983) (cit. on p. 85).

[BF10] Adrian Balint and Andreas Fröhlich. “Improving Stochastic Local Search for SAT with a New Probability Distribution”. In: *Theory and Applications of Satisfiability Testing – SAT 2010*. Ed. by Ofer Strichman and Stefan Szeider. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 10–15. ISBN: 978-3-642-14186-7 (cit. on p. 62).

[BHG09] Adrian Balint, Michael Henn, and Oliver Gableske. “A Novel Approach to Combine a SLS- and a DPLL-Solver for the Satisfiability Problem”. In: *Theory and Applications of Satisfiability Testing - SAT 2009*. Ed. by Oliver Kullmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 284–297. ISBN: 978-3-642-02777-2 (cit. on p. 66).

[BS12] Adrian Balint and Uwe Schöning. “Choosing Probability Distributions for Stochastic Local Search and the Role of Make versus Break”. In: *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing*. SAT’12. Trento, Italy: Springer-Verlag, 2012, pp. 16–29. ISBN: 9783642316111. DOI: 10.1007/978-3-642-31612-8_3 (cit. on p. 62).

[Bal+21] Tomas Balyo, Nils Froleyks, Marijn Heule, et al. *The Results of SAT Competition 2021*. Presented at SAT 2021 Conference, Barcelona (Spain). 2021. URL: https://satcompetition.github.io/2021/slides/ISC2021-fixed.pdf (cit. on p. 133).

[Bal+14] Tomás Balyo, Andreas Fröhlich, Marijn Heule, et al. “Everything You Always Wanted to Know about Blocked Sets (But Were Afraid to Ask)”. In: *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*. Ed. by Carsten Sinz and Uwe Egly. Vol. 8561. Lecture Notes in Computer Science. Springer, 2014, pp. 317–332. DOI: 10.1007/978-3-319-09284-3_24 (cit. on p. 58).

[Bal+20] Tomáš Balyo, Nils Froleyks, Marijn Heule, et al., eds. *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*. Vol. B-2020-1. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, 2020. URL: https://helda.helsinki.fi/bitstream/handle/10138/318450/sc2020_proceedings.pdf (cit. on p. 128).

[Bal+22] Tomáš Balyo, Marijn Heule, Markus Iser, et al., eds. *Proceedings of SAT Competition 2022: Solver and Benchmark Descriptions*. Vol. B-2022-1. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, 2022. URL: http://hdl.handle.net/10138/347211 (cit. on p. 132).

[BSS15] Tomáš Balyo, Peter Sanders, and Carsten Sinz. “HordeSat: A Massively Parallel Portfolio SAT Solver”. In: *Theory and Applications of Satisfiability Testing – SAT 2015*. Ed. by Marijn Heule and Sean Weaver. Cham: Springer International Publishing, 2015, pp. 156–172. ISBN: 978-3-319-24318-4 (cit. on p. 59).

[BS18] Tomáš Balyo and Carsten Sinz. “Parallel Satisfiability”. In: *Handbook of Parallel Constraint Reasoning*. Ed. by Youssef Hamadi and Lakhdar Sais. Cham: Springer International Publishing, 2018, pp. 3–29. ISBN: 978-3-319-63516-3. DOI: 10.1007/978-3-319-63516-3_1 (cit. on pp. 59, 66).

[BR99] Nikhil Bansal and Venkatesh Raman. “Upper Bounds for MaxSat: Further Improved”. In: *Algorithms and Computation, 10th International Symposium, ISAAC’99, Chennai, India, December 16-18, 1999, Proceedings*. Ed. by Alok Aggarwal and C. Pandu Rangan. Vol. 1741. Lecture Notes in Computer Science. Springer, 1999, pp. 247–258. DOI: 10.1007/3-540-46632-0_26 (cit. on pp. 90, 95).

[BM00] Luís Baptista and João Marques-Silva. “Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability”. In: *Principles and Practice of Constraint Programming – CP 2000*. Ed. by Rina Dechter. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 489–494. ISBN: 978-3-540-45349-9 (cit. on p. 57).

[BS97] Roberto J. Bayardo and Robert C. Schrag. “Using CSP Look-Back Techniques to Solve Real-World SAT Instances”. In: AAAI’97/IAAI’97. Providence, Rhode Island: AAAI Press, 1997, pp. 203–208. ISBN: 0262510952 (cit. on p. 46).

[Bea+96] Paul Beame, Russell Impagliazzo, Jan Krajíček, et al. “Lower Bounds on Hilbert’s Nullstellensatz and Propositional Proofs”. In: *Proceedings of the London Mathematical Society* s3-73.1 (1996), pp. 1–26. DOI: https://doi.org/10.1112/plms/s3-73.1.1 (cit. on p. 70).

[BKS04] Paul Beame, Henry Kautz, and Ashish Sabharwal. “Towards Understanding and Harnessing the Potential of Clause Learning”. In: *J. Artif. Int. Res.* 22.1 (2004), pp. 319–351. ISSN: 1076-9757 (cit. on p. 56).

[BP01] Paul Beame and Toniann Pitassi. “Propositional Proof Complexity: Past, Present, and Future”. In: *Current Trends in Theoretical Computer Science: Entering the 21st Centuary*. USA: World Scientific Publishing Co., Inc., 2001, pp. 42–70. ISBN: 9810244738 (cit. on p. 71).

[Bee+94] Antje Beeringer, Gerd Aschemann, Holger H. Hoos, et al. “GSAT versus Simulated Annealing”. In: *Proceedings of the Eleventh European Conference on Artificial Intelligence, Amsterdam, The Netherlands, August 8-12, 1994*. Ed. by Anthony G. Cohn. John Wiley and Sons, Chichester, 1994, pp. 130–134 (cit. on p. 66).

[BMM13] Anton Belov, António Morgado, and João Marques-Silva. “SAT-Based Preprocessing for MaxSAT”. In: *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*. Ed. by Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov. Vol. 8312. Lecture Notes in Computer Science. Springer, 2013, pp. 96–111. DOI: 10.1007/978-3-642-45221-5_7 (cit. on p. 91).

[BIW04] Eli Ben-Sasson, Russell Impagliazzo, and Avi Wigderson. “Near Optimal Separation Of Tree-Like And General Resolution”. In: *Comb.* 24.4 (2004), pp. 585–603. DOI: 10.1007/s00493-004-0036-5 (cit. on p. 73).

[BBP20a] Jeremias Berg, Fahiem Bacchus, and Alex Poole. “Abstract Cores in Implicit Hitting Set MaxSat Solving”. In: *Theory and Applications of Satisfiability Testing – SAT 2020*. Ed. by Luca Pulina and Martina Seidl. Cham: Springer International Publishing, 2020, pp. 277–294. ISBN: 978-3-030-51825-7.

[BBP20b] Jeremias Berg, Fahiem Bacchus, and Alex Poole. “Abstract Cores in Implicit Hitting Set MaxSat Solving”. In: *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*. Ed. by Luca Pulina and Martina Seidl. Vol. 12178. Lecture Notes in Computer Science. Springer, 2020, pp. 277–294. DOI: 10.1007/978-3-030-51825-7_20 (cit. on p. 110).

[BHJ18] Jeremias Berg, Antti Hyttinen, and Matti Järvisalo. “Applications of MaxSAT in Data Analysis”. In: *Proceedings of Pragmatics of SAT 2015, Austin, Texas, USA, September 23, 2015 / Pragmatics of SAT 2018, Oxford, UK, July 7, 2018*. Ed. by Daniel Le Berre and Matti Järvisalo. Vol. 59. EPiC Series in Computing. EasyChair, 2018, pp. 50–64. DOI: 10.29007/3qkh (cit. on p. 85).

[BJ13] Jeremias Berg and Matti Järvisalo. “Optimal Correlation Clustering via MaxSAT”. In: *13th IEEE International Conference on Data Mining Workshops, ICDM Workshops, TX, USA, December 7-10, 2013*. Ed. by Wei Ding, Takashi Washio, Hui Xiong, et al. IEEE Computer Society, 2013, pp. 750–757. DOI: 10.1109/ICDMW.2013.99 (cit. on p. 85).

[BSJ15a] Jeremias Berg, Paul Saikko, and Matti Järvisalo. “Improving the Effectiveness of SAT-Based Preprocessing for MaxSAT”. In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Ed. by Qiang Yang and Michael J. Wooldridge. AAAI Press, 2015, pp. 239–245 (cit. on p. 91).

[BSJ15b] Jeremias Berg, Paul Saikko, and Matti Järvisalo. “Re-using Auxiliary Variables for MaxSAT Preprocessing”. In: *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*. IEEE Computer Society, 2015, pp. 813–820. DOI: 10.1109/ICTAI.2015.120 (cit. on p. 91).

[BP10] Daniel Le Berre and Anne Parrain. “The Sat4j library, release 2.2”. In: *J. Satisf. Boolean Model. Comput.* 7.2-3 (2010), pp. 59–6. DOI: 10.3233/sat190075 (cit. on pp. 101, 102, 104).

[BDS11] Pierre Bieber, Remi Delmas, and Christel Seguin. “DALculus - Theory and Tool for Development Assurance Level Allocation”. In: *Computer Safety, Reliability, and Security - 30th International Conference, SAFECOMP 2011, Naples, Italy, September 19-22, 2011. Proceedings*. Ed. by Francesco Flammini, Sandro Bologna, and Valeria Vittorini. Vol. 6894. Lecture Notes in Computer Science. Springer, 2011, pp. 43–56. DOI: 10.1007/978-3-642-24270-0_4 (cit. on p. 85).

[BHM21] A. Biere, M. Heule, and H. van Maaren. *Handbook of Satisfiability: Second Edition*. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021. ISBN: 9781643681610. URL: https://books.google.fr/books?id=dUAvEAAAQBAJ (cit. on pp. 18, 39).

[Bie08a] Armin Biere. “Adaptive Restart Strategies for Conflict Driven SAT Solvers”. In: *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing*. SAT’08. Guangzhou, China: Springer-Verlag, 2008, pp. 28–33. ISBN: 3540797181 (cit. on pp. 53, 57, 122).

[Bie08b] Armin Biere. “PicoSAT Essentials”. In: *J. Satisf. Boolean Model. Comput.* 4 (2008), pp. 75–97 (cit. on pp. 42, 47).

[Bie10a] Armin Biere. “Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010”. en. In: (2010). DOI: 10.35011/FMVTR.2010-1 (cit. on p. 59).

[Bie10b] Armin Biere. “Lingeling, plingeling, picosat and precosat at sat race 2010”. In: *Technical Report 10/1*. FMV Reports Series, Institute for Formal Models and Verification, Johannes Kepler University. Linz, Austria, 2010 (cit. on pp. 42, 53).

[Bie12a] Armin Biere. “Lingeling and friends entering the SAT Challenge 2012”. In: Department of Computer Science Report Series B B-2012-2 (2012). Ed. by A Balint, A Belov, D Diepold, et al., pp. 33–34 (cit. on p. 59).

[Bie12b] Armin Biere. “Preprocessing and Inprocessing Techniques in SAT”. In: *Hardware and Software: Verification and Testing*. Ed. by Kerstin Eder, João Lourenço, and Onn Shehory. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–1. ISBN: 978-3-642-34188-5. URL: http://www.kr.tuwien.ac.at/drm/ordyniak/worker/slides/Armin-Biere.pdf (cit. on p. 58).

[Bie17] Armin Biere. “CaDiCaL, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2017”. In: *Proc. of SAT Competition 2017 – Solver and Benchmark Descriptions*. Ed. by Tomáš Balyo, Marijn Heule, and Matti Järvisalo. Vol. B-2017-1. Department of Computer Science Series of Publications B. University of Helsinki, 2017, pp. 14–15 (cit. on pp. 42, 52, 122).

[Bie21] Armin Biere. “Bounded Model Checking”. In: *Handbook of Satisfiability - Second Edition*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, et al. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 739–764. DOI: 10.3233/FAIA201002 (cit. on p. 39).

[Bie+20] Armin Biere, Katalin Fazekas, Mathias Fleury, et al. “CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2020”. In: *Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions*. Ed. by Tomas Balyo, Nils Froleyks, Marijn Heule, et al. Vol. B-2020-1. Department of Computer Science Report Series B. University of Helsinki, 2020, pp. 51–53 (cit. on pp. 42, 52, 59, 122, 124).

[BF20] Armin Biere and Mathias Fleury. “Chasing target phases”. In: *Proceedings of Pragmatics of (SAT)* (2020) (cit. on pp. 57, 59).

[BFH21] Armin Biere, Mathias Fleury, and Maximillian Heisinger. “CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling Entering the SAT Competition 2021”. In: *Proceedings of SAT Competition 2021: Solver and Benchmark Descriptions*. Ed. by Tomas Balyo and Nils Froleyks and Marijn Heule and Markus Iser and Matti Järvisalo and Martin Suda. Vol. B-2021-1. Department of Computer Science Report Series B. University of Helsinki, 2021 (cit. on p. 132).

[BF19] Armin Biere and Andreas Fr\"ohlich. “Evaluating CDCL Restart Schemes”. In: *Proceedings of Pragmatics of SAT 2015 and 2018*. Ed. by Daniel Le Berre and Matti J\"arvisalo. Vol. 59. EPiC Series in Computing. EasyChair, 2019, pp. 1–17. DOI: 10.29007/89dw (cit. on p. 57).

[BF15] Armin Biere and Andreas Fröhlich. “Evaluating CDCL Variable Scoring Schemes”. In: *Theory and Applications of Satisfiability Testing – SAT 2015*. Ed. by Marijn Heule and Sean Weaver. Cham: Springer International Publishing, 2015, pp. 405–422. ISBN: 978-3-319-24318-4 (cit. on pp. 53, 55, 122).

[BTS19] Paul Maximilian Bittner, Thomas Thüm, and Ina Schaefer. “SAT Encodings of the At-Most-k Constraint - A Case Study on Configuring University Courses”. In: *Software Engineering and Formal Methods - 17th International Conference, SEFM 2019, Oslo, Norway, September 18-20, 2019, Proceedings*. Ed. by Peter Csaba Ölveczky and Gwen Salaün. Vol. 11724. Lecture Notes in Computer Science. Springer, 2019, pp. 127–144. DOI: 10.1007/978-3-030-30446-1_7 (cit. on p. 100).

[Blä+22] Thomas Bläsius, Tobias Friedrich, David Stangl, et al. “An Efficient Branch-and-Bound Solver for Hitting Set”. In: *Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2022, Alexandria, VA, USA, January 9-10, 2022*. Ed. by Cynthia A. Phillips and Bettina Speckmann. SIAM, 2022, pp. 209–220. DOI: 10.1137/1.9781611977042.17 (cit. on p. 110).

[Bof+15] Miquel Bofill, Marc Garcia, Josep Suy, et al. “MaxSAT-Based Scheduling of B2B Meetings”. In: *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*. Ed. by Laurent Michel. Vol. 9075. Lecture Notes in Computer Science. Springer, 2015, pp. 65–73. DOI: 10.1007/978-3-319-18008-3_5 (cit. on p. 85).

[BS96] Max Böhm and Ewald Speckenmeyer. “A fast parallel SAT-solver — efficient workload balancing”. In: *Annals of Mathematics and Artificial Intelligence* 17.2 (1996), pp. 381–400. DOI: 10.1007/bf02127976 (cit. on p. 59).

[Bon+18] Maria Luisa Bonet, Sam Buss, Alexey Ignatiev, et al. “MaxSAT Resolution With the Dual Rail Encoding”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 6565–6572. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16782 (cit. on p. 113).

[Bon+98] Maria Luisa Bonet, Juan Luis Esteban, Nicola Galesi, et al. “Exponential Separations between Restricted Resolution and Cutting Planes Proof Systems”. In: *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*. FOCS ’98. USA: IEEE Computer Society, 1998, p. 638. ISBN: 0818691727 (cit. on p. 73).

[BL20] Maria Luisa Bonet and Jordi Levy. “Equivalence Between Systems Stronger Than Resolution”. In: *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*. Ed. by Luca Pulina and Martina Seidl. Vol. 12178. Lecture Notes in Computer Science. Springer, 2020, pp. 166–181. DOI: 10.1007/978-3-030-51825-7_13 (cit. on pp. 20, 113, 115, 151, 177).

[BLM06] Maria Luisa Bonet, Jordi Levy, and Felip Manyà. “A Complete Calculus for Max-SAT”. In: *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*. Ed. by Armin Biere and Carla P. Gomes. Vol. 4121. Lecture Notes in Computer Science. Springer, 2006, pp. 240–251. DOI: 10.1007/11814948_24 (cit. on pp. 89, 106, 113–116, 151, 169, 170, 181).

[BLM07] Maria Luisa Bonet, Jordi Levy, and Felip Manyà. “Resolution for Max-SAT”. In: *Artif. Intell.* 171.8-9 (2007), pp. 606–618. DOI: 10.1016/j.artint.2007.03.001 (cit. on pp. 20, 89, 91, 106, 113–116, 151, 152, 169, 170, 181).

[Boo54] George Boole. *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities*. Dover, 1854 (cit. on pp. 18, 33).

[BBJ07] G.S. Boolos, J.P. Burgess, and R.C. Jeffrey. *Computability and Logic*. Cambridge University Press, 2007. ISBN: 9780521877527. URL: https://books.google.fr/books?id=kKPA1DFg44C (cit. on p. 27).

[BF98] Brian Borchers and Judith Furman. “A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems”. In: *Journal of Combinatorial Optimization* 2.4 (1998), pp. 299–306. DOI: 10.1023/a:1009725216438 (cit. on pp. 86, 87, 89).

[BHS94] Endre Boros, Peter L. Hammer, and Xiaorong Sun. “Recognition of q-Horn formulae in linear time”. In: *Discrete Applied Mathematics* 55.1 (1994), pp. 1–13. ISSN: 0166-218X. DOI: 10.1016/0166-218X(94)90033-7 (cit. on pp. 38, 67).

[Bou+18] Abdelhamid Boudane, Saïd Jabbour, Lakhdar Sais, et al. “SAT-Based Data Mining”. In: *International Journal on Artificial Intelligence Tools* 27.01 (2018), p. 1840002. DOI: 10.1142/S021821301840002X (cit. on p. 39).

[BD05] Dalila Boughaci and Habiba Drias. “Efficient and Experimental Meta-heuristics for MAX-SAT Problems”. In: *Experimental and Efficient Algorithms*. Ed. by Sotiris E. Nikoletseas. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 501–512. ISBN: 978-3-540-32078-4 (cit. on p. 111).

[BBG12] Djallel Bouneffouf, Amel Bouzeghoub, and Alda Lopes Gançarski. “A Contextual-Bandit Algorithm for Mobile Context-Aware Recommender System”. In: *Neural Information Processing*. Ed. by Tingwen Huang, Zhigang Zeng, Chuandong Li, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 324–331. ISBN: 978-3-642-34487-9 (cit. on p. 77).

[Bou+19] Djallel Bouneffouf, Srinivasan Parthasarathy, Horst Samulowitz, et al. “Optimal Exploitation of Clustering and History Information in Multi-Armed Bandit”. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI’19. Macao, China: AAAI Press, 2019, pp. 2016–2022. ISBN: 9780999241141 (cit. on p. 79).

[BRA20] Djallel Bouneffouf, Irina Rish, and Charu Aggarwal. “Survey on applications of multi-armed and contextual bandits”. In: *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2020, pp. 1–8 (cit. on p. 80).

[BCC94] D. Bovet, P. Crescenzi, and P. Crescenzl. *Introduction to the Theory of Complexity*. Prentice-Hall international series in civil engineering and engineering mechanics. Prentice Hall, 1994. ISBN: 9780139153808. URL: https://books.google.fr/books?id=xucmAQAAIAAJ (cit. on p. 31).

[BH22] Guy Bresler and Brice Huang. “The algorithmic phase transition of random k-sat for low degree polynomials”. In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2022, pp. 298–309. DOI: 10.1109/FOCS52979.2021.00038 (cit. on p. 66).

[BSK11] Piotr Brodka, Pawel Stawiak, and Przemyslaw Kazienko. “Shortest Path Discovery in the Multilayered Social Network”. In: *2011 International Conference on Advances in Social Networks Analysis and Mining*. 2011, pp. 497–501. DOI: 10.1109/ASONAM.2011.67 (cit. on p. 25).

[BC12] Sébastien Bubeck and Nicolò Cesa-Bianchi. “Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems”. In: *Found. Trends Mach. Learn.* 5.1 (2012), pp. 1–122. DOI: 10.1561/2200000024.

[BP07a] Joshua Buresh-Oppenheim and Toniann Pitassi. “The Complexity of Resolution Refinements”. In: *J. Symb. Log.* 72.4 (2007), pp. 1336–1352. DOI: 10.2178/jsl/1203350790 (cit. on pp. 71, 169).

[BP07b] Joshua Buresh-Oppenheim and Toniann Pitassi. “The Complexity of Resolution Refinements”. In: *J. Symb. Log.* 72.4 (2007), pp. 1336–1352. DOI: 10.2178/jsl/1203350790 (cit. on pp. 72, 74).

[BB92] Michael Buro and H Kleine Büning. *Report on a SAT competition*. Fachbereich Math.-Informatik, Univ. Gesamthochschule Paderborn, Germany, 1992 (cit. on p. 49).

[Bus15] Sam Buss. “Propositional Proofs in Frege and Extended Frege Systems (Abstract)”. In: *Computer Science – Theory and Applications*. Ed. by Lev D. Beklemishev and Daniil V. Musatov. Cham: Springer International Publishing, 2015, pp. 1–6. ISBN: 978-3-319-20297-6 (cit. on p. 70).

[BJ16] Sam Buss and Jan Johannsen. “On Linear Resolution”. In: *J. Satisf. Boolean Model. Comput.* 10.1 (2016), pp. 23–35. DOI: 10.3233/sat190112 (cit. on pp. 72, 74).

[BN21a] Sam Buss and Jakob Nordström. “Proof Complexity and SAT Solving”. In: *Handbook of Satisfiability* 336 (2021), pp. 233–350 (cit. on p. 71).

[BN21b] Sam Buss and Jakob Nordström. “Proof Complexity and SAT Solving.” In: *Handbook of Satisfiability* 336 (2021), pp. 233–350 (cit. on p. 71).

[Bus12] Samuel R. Buss. “Towards NP–P via proof complexity and search”. In: *Annals of Pure and Applied Logic* 163.7 (2012). The Symposium on Logical Foundations of Computer Science 2009, pp. 906–917. ISSN: 0168-0072. DOI: https://doi.org/10.1016/j.apal.2011.09.009 (cit. on p. 71).

[CJS15] Shaowei Cai, Zhong Jie, and Kaile Su. “An effective variable selection heuristic in SLS for weighted Max-2-SAT”. In: *J. Heuristics* 21.3 (2015), pp. 433–456. DOI: 10.1007/s10732-015-9284-3 (cit. on p. 111).

[CL20] Shaowei Cai and Zhendong Lei. “Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability”. In: *Artif. Intell.* 287 (2020), p. 103354. DOI: 10.1016/j.artint.2020.103354 (cit. on pp. 111, 112).

[Cai+16] Shaowei Cai, Chuan Luo, Jinkun Lin, et al. “New local search methods for partial MaxSAT”. In: *Artif. Intell.* 240 (2016), pp. 1–18. DOI: 10.1016/j.artint.2016.07.006 (cit. on pp. 111, 112).

[CLS14] Shaowei Cai, Chuan Luo, and Kaile Su. “Scoring Functions Based on Second Level Score for k-SAT with Long Clauses”. In: *J. Artif. Intell. Res.* 51 (2014), pp. 413–441. DOI: 10.1613/jair.4480 (cit. on p. 64).

[CLS15a] Shaowei Cai, Chuan Luo, and Kaile Su. “CCAnr: A Configuration Checking Based Local Search Solver for Non-random Satisfiability”. In: *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*. Ed. by Marijn Heule and Sean A. Weaver. Vol. 9340. Lecture Notes in Computer Science. Springer, 2015, pp. 1–8. DOI: 10.1007/978-3-319-24318-4_1 (cit. on pp. 62, 64–66).

[CLS15b] Shaowei Cai, Chuan Luo, and Kaile Su. “Improving WalkSAT By Effective Tie-Breaking and Efficient Implementation”. In: *Comput. J.* 58.11 (2015), pp. 2864–2875. DOI: 10.1093/comjnl/bxu135 (cit. on p. 62).

[Cai+14] Shaowei Cai, Chuan Luo, John Thornton, et al. “Tailoring Local Search for Partial MaxSAT”. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*. Ed. by Carla E. Brodley and Peter Stone. AAAI Press, 2014, pp. 2623–2629. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8141 (cit. on pp. 111, 112).

[Cai+21] Shaowei Cai, Chuan Luo, Xindi Zhang, et al. “Improving Local Search for Structured SAT Formulas via Unit Propagation Based Construct and Cut Initialization”. In: *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Ed. by Laurent D. Michel. Vol. 210. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 5:1–5:10. ISBN: 978-3-95977-211-2. DOI: 10.4230/LIPIcs.CP.2021.5 (cit. on p. 66).

[CS11] Shaowei Cai and Kaile Su. “Local Search with Configuration Checking for SAT”. In: *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011*. IEEE Computer Society, 2011, pp. 59–66. DOI: 10.1109/ICTAI.2011.18 (cit. on pp. 62–64).

[CS12] Shaowei Cai and Kaile Su. “Configuration Checking with Aspiration in Local Search for SAT”. In: *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. Ed. by Jörg Hoffmann and Bart Selman. AAAI Press, 2012. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4966 (cit. on pp. 62, 64).

[CS13a] Shaowei Cai and Kaile Su. “Comprehensive Score: Towards Efficient Local Search for SAT with Long Clauses”. In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. Ed. by Francesca Rossi. IJCAI/AAAI, 2013, pp. 489–495. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6206 (cit. on p. 64).

[CS13b] Shaowei Cai and Kaile Su. “Local search for Boolean Satisfiability with configuration checking and subscore”. In: *Artif. Intell.* 204 (2013), pp. 75–98. DOI: 10.1016/j.artint.2013.09.001 (cit. on pp. 62, 64, 65).

[CSL13] Shaowei Cai, Kaile Su, and Chuan Luo. “Improving WalkSAT for Random k-Satisfiability Problem with k > 3”. In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. Ed. by Marie desJardins and Michael L. Littman. AAAI Press, 2013. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6145 (cit. on p. 62).

[CZ21a] Shaowei Cai and Xindi Zhang. “Deep Cooperation of CDCL and Local Search for SAT”. In: *Theory and Applications of Satisfiability Testing – SAT 2021*. Ed. by Chu-Min Li and Felip Manyà. Cham: Springer International Publishing, 2021, pp. 64–81. ISBN: 978-3-030-80223-3.

[CZ21b] Shaowei Cai and Xindi Zhang. “Deep Cooperation of CDCL and Local Search for SAT”. In: *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*. Ed. by Chu-Min Li and Felip Manyà. Vol. 12831. Lecture Notes in Computer Science. Springer, 2021, pp. 64–81. DOI: 10.1007/978-3-030-80223-3_6 (cit. on p. 59).

[Cai+22] Shaowei Cai, Xindi Zhang, Mathias Fleury, et al. “Better Decision Heuristics in CDCL through Local Search and Target Phases”. In: *J. Artif. Intell. Res.* 74 (2022), pp. 1515–1563. DOI: 10.1613/jair.1.13666 (cit. on p. 59).

[CSJ20] Henrik E. C. Cao, Riku Sarlin, and Alexander Jung. “Learning Explainable Decision Rules via Maximum Satisfiability”. In: *IEEE Access* 8 (2020), pp. 218180–218185. DOI: 10.1109/ACCESS.2020.3041040 (cit. on p. 85).

[CZ12] Renato Carmo and Alexandre Prusch Züge. “Branch and bound algorithms for the maximum clique problem under a unified framework”. In: *J. Braz. Comput. Soc.* 18.2 (2012), pp. 137–151. DOI: 10.1007/s13173-011-0050-6 (cit. on p. 85).

[Cha+97] Byungki Cha, Kazuo Iwama, Yahiko Kambayashi, et al. “Local Search Algorithms for Partial MaxSAT”. In: *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Conference on Innovative Applications of Artificial Intelligence*. AAAI’97/IAAI’97. Providence, Rhode Island: AAAI Press, 1997, pp. 263–268. ISBN: 0-262-51095-2 (cit. on p. 85).

[Cha99] Jean-Luc Chabert. “Algorithms for Arithmetic Operations”. In: *A History of Algorithms: From the Pebble to the Microchip*. Ed. by Jean-Luc Chabert. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 7–47. ISBN: 978-3-642-18192-4. DOI: 10.1007/978-3-642-18192-4_2 (cit. on p. 26).

[Cha+11] Karthekeyan Chandrasekaran, Richard M. Karp, Erick Moreno-Centeno, et al. “Algorithms for Implicit Hitting Set Problems”. In: *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*. Ed. by Dana Randall. SIAM, 2011, pp. 614–629. DOI: 10.1137/1.9781611973082.48 (cit. on p. 109).

[CH91] V. Chandru and J. N. Hooker. “Extended Horn Sets in Propositional Logic”. In: *J. ACM* 38.1 (1991), pp. 205–221. ISSN: 0004-5411. DOI: 10.1145/102782.102789 (cit. on pp. 38, 67).

[CWX17] Wenjing Chang, Guanfeng Wu, and Yang Xu. “Adding a LBD-based rewarding mechanism in branching heuristic for SAT solvers”. In: *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*. 2017, pp. 1–6. DOI: 10.1109/ISKE.2017.8258780 (cit. on p. 55).

[CXC19] Wenjing Chang, Yang Xu, and Shuwei Chen. “A New Rewarding Mechanism for Branching Heuristic in SAT Solvers”. In: *International Journal of Computational Intelligence Systems* 12 (1 2019), pp. 334–341. ISSN: 1875-6883. DOI: https://doi.org/10.2991/ijcis.2019.125905649 (cit. on p. 55).

[Che+10] Yibin Chen, Sean Safarpour, João Marques-Silva, et al. “Automated Design Debugging With Maximum Satisfiability”. In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 29.11 (2010), pp. 1804–1817. DOI: 10.1109/TCAD.2010.2061270 (cit. on p. 85).

[Che+09] Yibin Chen, Sean Safarpour, Andreas G. Veneris, et al. “Spatial and temporal design debug using partial MaxSAT”. In: *Proceedings of the 19th ACM Great Lakes Symposium on VLSI 2009, Boston Area, MA, USA, May 10-12 2009*. Ed. by Fabrizio Lombardi, Sanjukta Bhanja, Yehia Massoud, et al. ACM, 2009, pp. 345–350. DOI: 10.1145/1531542.1531621 (cit. on p. 85).

[CH19] Mohamed Sami Cherif and Djamal Habet. “Towards the Characterization of Max-Resolution Transformations of UCSs by UP-Resilience”. In: *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings*. Ed. by Thomas Schiex and Simon de Givry. Vol. 11802. Lecture Notes in Computer Science. Springer, 2019, pp. 91–107. DOI: 10.1007/978-3-030-30048-7_6 (cit. on p. 114).

[CHA20] Mohamed Sami Cherif, Djamal Habet, and André Abramé. “Understanding the Power of Max-SAT Resolution through UP-Resilience”. In: *Artif. Intell.* 289 (2020), p. 103397. DOI: 10.1016/j.artint.2020.103397 (cit. on pp. 3, 20, 136).

[CHA21] Mohamed Sami Cherif, Djamal Habet, and André Abramé. *Understanding the power of Max-SAT resolution through UP-resilience*. Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21). Poster. Aug. 2021. URL: https://hal-amu.archives-ouvertes.fr/hal-03334479 (cit. on pp. 3, 20, 136).

[CHP22a] Mohamed Sami Cherif, Djamal Habet, and Matthieu Py. “De la résolution à la max-résolution”. In: *Journées Francophones de Programmation par Contraintes (JFPC)*. Saint-Étienne, France, June 2022 (cit. on pp. 4, 20, 153).

[CHP22b] Mohamed Sami Cherif, Djamal Habet, and Matthieu Py. “From Crossing-Free Resolution to Max-SAT Resolution”. In: *28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel*. Ed. by Christine Solnon. Vol. 235. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 12:1–12:17. DOI: 10.4230/LIPIcs.CP.2022.12 (cit. on pp. 3, 20, 153).

[CHT20] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “On the Refinement of Conflict History Search Through Multi-Armed Bandit”. In: *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*. IEEE, 2020, pp. 264–271. DOI: 10.1109/ICTAI50040.2020.00050 (cit. on pp. 4, 20, 133, 180).

[CHT21a] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “Combining VSIDS and CHB Using Restarts in SAT”. In: *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*. Ed. by Laurent D. Michel. Vol. 210. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 20:1–20:19. DOI: 10.4230/LIPIcs.CP.2021.20 (cit. on pp. 3, 20, 120, 180).

[CHT21b] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “Kissat_MAB: Combining VSIDS and CHB through Multi-Armed Bandit”. In: *Proceedings of SAT Competition 2021: Solver and Benchmark Descriptions*. Ed. by Tomas Balyo and Nils Froleyks and Marijn Heule and Markus Iser and Matti Järvisalo and Martin Suda. Vol. B-2021-1. Department of Computer Science Series of Publications B. University of Helsinki, 2021, p. 15 (cit. on pp. 20, 132, 180).

[CHT21c] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “Raffiner l’heuristique CHS à l’aide de bandits”. In: *Journées Francophones de Programmation par Contraintes (JFPC)*. Online, France, June 2021 (cit. on pp. 4, 20, 133, 180).

[CHT21d] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “Un bandit manchot pour combiner CHB et VSIDS”. In: *Journées Francophones de Programmation par Contraintes (JFPC)*. Online, France, June 2021 (cit. on pp. 4, 20, 120, 180).

[CHT21e] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “Verifying Optimums of (Partial) Max-SAT Formulas”. In: Department of Computer Science Series of Publications B B-2021-1 (2021). Ed. by Tomas Balyo and Nils Froleyks and Marijn Heule and Markus Iser and Matti Järvisalo and Martin Suda, p. 49 (cit. on p. 20).

[CHT22a] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “Kissat MAB: Upper Confidence Bound Strategies to Combine VSIDS and CHB”. In: *SAT COMPETITION 2022*. Department of Computer Science Series of Publications B B-2022-1 (2022). Ed. by Tomas Balyo and Marijn Heule and Markus Iser and Matti Järvisalo and Martin Suda, p. 14 (cit. on pp. 20, 132, 180).

[CHT22b] Mohamed Sami Cherif, Djamal Habet, and Cyril Terrioux. “Verifying Optimums of Weighted (Partial) Max-SAT Formulas”. In: Department of Computer Science Series of Publications B B-2022-1 (2022). Ed. by Tomas Balyo and Marijn Heule and Markus Iser and Matti Järvisalo and Martin Suda, p. 59 (cit. on p. 20).

[CMY19] Md Solimul Chowdhury, Martin Müller, and Jia-Huai You. “Exploiting Glue Clauses to Design Effective CDCL Branching Heuristics”. In: *Principles and Practice of Constraint Programming*. Ed. by Thomas Schiex and Simon de Givry. Cham: Springer International Publishing, 2019, pp. 126–143. ISBN: 978-3-030-30048-7 (cit. on p. 55).

[Chu+11a] Wei Chu, Lihong Li, Lev Reyzin, et al. “Contextual Bandits with Linear Payoff Functions”. In: *Proceedings of International Conference on Artificial Intelligence and Statistics*. 2011, pp. 208–214 (cit. on p. 134).

[Chu+11b] Wei Chu, Lihong Li, Lev Reyzin, et al. “Contextual Bandits with Linear Payoff Functions”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*. Ed. by Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík. Vol. 15. JMLR Proceedings. JMLR.org, 2011, pp. 208–214. URL: http://proceedings.mlr.press/v15/chu11a/chu11a.pdf (cit. on p. 79).

[Chu36] Alonzo Church. “An Unsolvable Problem of Elementary Number Theory”. In: *American Journal of Mathematics* 58.2 (1936), pp. 345–363. ISSN: 00029327, 10806377. URL: https://www.jstor.org/stable/2371045 (cit. on p. 27).

[Chv85] V. Chvátal. “Cutting Planes in Combinatorics”. In: *European Journal of Combinatorics* 6.3 (1985), pp. 217–226. ISSN: 0195-6698. DOI: https://doi.org/10.1016/S0195-6698(85)80031-7 (cit. on p. 70).

[CEI96] Matthew Clegg, Jeffery Edmonds, and Russell Impagliazzo. “Using the Groebner Basis Algorithm to Find Proofs of Unsatisfiability”. In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC ’96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 174–183. ISBN: 0897917855. DOI: 10.1145/237814.237860 (cit. on p. 71).

[Cob65] Alan Cobham. “The Intrinsic Computational Difficulty of Functions”. In: *Logic, Methodology and Philosophy of Science: Proceedings of the 1964 International Congress (Studies in Logic and the Foundations of Mathematics)*. Ed. by Yehoshua Bar-Hillel. North-Holland Publishing, 1965, pp. 24–30 (cit. on p. 31).

[CLS08] Michael Codish, Vitaly Lagoon, and Peter J. Stuckey. “Logic programming with satisfiability”. In: *Theory Pract. Log. Program.* 8.1 (2008), pp. 121–128. DOI: 10.1017/S1471068407003146 (cit. on p. 102).

[CS18] Dor Cohen and Ofer Strichman. “The Impact of Entropy and Solution Density on Selected SAT Heuristics”. In: *Entropy* 20.9 (2018). ISSN: 1099-4300. DOI: 10.3390/e20090713 (cit. on pp. 67, 69).

[CP16] Amin Coja-Oghlan and Konstantinos Panagiotou. “The asymptotic k-SAT threshold”. In: *Advances in Mathematics* 288 (2016), pp. 985–1068. ISSN: 0001-8708. DOI: https://doi.org/10.1016/j.aim.2015.11.007 (cit. on p. 66).

[Coo71] Stephen A. Cook. “The Complexity of Theorem-Proving Procedures”. In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*. Ed. by Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman. ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047 (cit. on pp. 18, 31, 38, 66).

[CR79] Stephen A. Cook and Robert A. Reckhow. “The relative efficiency of propositional proof systems”. In: *Journal of Symbolic Logic* 44.1 (1979), pp. 36–50. DOI: 10.2307/2273702 (cit. on p. 71).

[CCT87] W. Cook, C.R. Coullard, and Gy. Turán. “On the complexity of cutting-plane proofs”. In: *Discrete Applied Mathematics* 18.1 (1987), pp. 25–38. ISSN: 0166-218X. DOI: https://doi.org/10.1016/0166-218X(87)90039-4 (cit. on p. 70).

[Cor+09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, et al. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. ISBN: 978-0-262-03384-8. URL: http://mitpress.mit.edu/books/introduction-algorithms (cit. on pp. 31, 52, 101).

[CA96] James M. Crawford and Larry D. Auton. “Experimental results on the crossover point in random 3-SAT”. In: *Artificial Intelligence* 81.1 (1996). Frontiers in Problem Solving: Phase Transitions and Complexity, pp. 31–57. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(95)00046-1 (cit. on p. 50).

[CO17] Nadia Creignou and Frédéric Olive. *Complexité, Master d’informatique - Semestre 1*. French. Lecture notes. 2017.

[DAg99] Marcello D’Agostino. “Tableau Methods for Classical Propositional Logic”. In: *Handbook of Tableau Methods*. Ed. by Marcello D’Agostino, Dov M. Gabbay, Reiner Hähnle, et al. Dordrecht: Springer Netherlands, 1999, pp. 45–123. ISBN: 978-94-017-1754-0. DOI: 10.1007/978-94-017-1754-0_2 (cit. on p. 71).

[Dav13a] Jessica Davies. “Solving MaxSAT by decoupling optimization and satisfaction”. PhD thesis. University of Toronto, 2013 (cit. on pp. 109, 110).

[DB11] Jessica Davies and Fahiem Bacchus. “Solving MAXSAT by Solving a Sequence of Simpler SAT Instances”. In: *Principles and Practice of Constraint Programming – CP 2011*. Ed. by Jimmy Lee. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 225–239. ISBN: 978-3-642-23786-7 (cit. on pp. 109, 110).

[DB13a] Jessica Davies and Fahiem Bacchus. “Exploiting the Power of mip Solvers in maxsat”. In: *Theory and Applications of Satisfiability Testing – SAT 2013*. Ed. by Matti Järvisalo and Allen Van Gelder. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 166–181. ISBN: 978-3-642-39071-5 (cit. on p. 110).

[DB13b] Jessica Davies and Fahiem Bacchus. “Postponing Optimization to Speed Up MAXSAT Solving”. In: *Principles and Practice of Constraint Programming*. Ed. by Christian Schulte. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 247–262. ISBN: 978-3-642-40627-0 (cit. on p. 110).

[DCB10a] Jessica Davies, Jeremy Cho, and Fahiem Bacchus. “Using Learnt Clauses in maxsat”. In: *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*. Ed. by David Cohen. Vol. 6308. Lecture Notes in Computer Science. Springer, 2010, pp. 176–190. DOI: 10.1007/978-3-642-15396-9_17 (cit. on p. 109).

[DCB10b] Jessica Davies, Jeremy Cho, and Fahiem Bacchus. “Using Learnt Clauses in maxsat”. In: *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*. Ed. by David Cohen. Vol. 6308. Lecture Notes in Computer Science. Springer, 2010, pp. 176–190. DOI: 10.1007/978-3-642-15396-9_17 (cit. on p. 91).

[Dav13b] M. Davis. *Computability and Unsolvability*. Dover Books on Computer Science. Dover Publications, 2013. ISBN: 9780486151069. URL: https://books.google.fr/books?id=nbOqAAAAQBAJ (cit. on p. 27).

[DLL62] Martin Davis, George Logemann, and Donald Loveland. “A Machine Program for Theorem-Proving”. In: *Commun. ACM* 5.7 (1962), pp. 394–397. ISSN: 0001-0782. DOI: 10.1145/368273.368557 (cit. on pp. 40, 46, 88, 89).

[DP60] Martin Davis and Hilary Putnam. “A Computing Procedure for Quantification Theory”. In: *J. ACM* 7.3 (1960), pp. 201–215. ISSN: 0004-5411. DOI: 10.1145/321033.321034 (cit. on pp. 40, 58, 71, 88, 89).

[De +03] Simon De Givry, Javier Larrosa, Pedro Meseguer, et al. “Solving Max-SAT as weighted CSP”. In: *Principles and Practice of Constraint Programming (CP)*. Springer. 2003, pp. 363–376 (cit. on p. 111).

[DM17] Emir Demirovic and Nysret Musliu. “MaxSAT-based large neighborhood search for high school timetabling”. In: *Comput. Oper. Res.* 78 (2017), pp. 172–180. DOI: 10.1016/j.cor.2016.08.004 (cit. on p. 85).

[DMW19] Emir Demirovic, Nysret Musliu, and Felix Winter. “Modeling and solving staff scheduling with partial weighted maxSAT”. In: *Ann. Oper. Res.* 275.1 (2019), pp. 79–99. DOI: 10.1007/s10479-017-2693-y (cit. on p. 85).

[DS19] Emir Demirović and Peter J. Stuckey. “Techniques Inspired by Local Search for Incomplete MaxSAT and the Linear Algorithm: Varying Resolution and Solution-Guided Search”. In: *Principles and Practice of Constraint Programming*. Ed. by Thomas Schiex and Simon de Givry. Cham: Springer International Publishing, 2019, pp. 177–194. ISBN: 978-3-030-30048-7 (cit. on p. 112).

[DD04] Gilles Dequen and Olivier Dubois. “kcnfs: An Efficient Solver for Random k-SAT Formulae”. In: *Theory and Applications of Satisfiability Testing*. Ed. by Enrico Giunchiglia and Armando Tacchella. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 486–501. ISBN: 978-3-540-24605-3 (cit. on pp. 51, 67).

[DGS07] Bistra Dilkina, Carla P. Gomes, and Ashish Sabharwal. “Tradeoffs in the Complexity of Backdoor Detection”. In: *Principles and Practice of Constraint Programming – CP 2007*. Ed. by Christian Bessière. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 256–270. ISBN: 978-3-540-74970-7 (cit. on p. 67).

[DGS09] Bistra Dilkina, Carla P. Gomes, and Ashish Sabharwal. “Backdoors in the Context of Learning”. In: *Theory and Applications of Satisfiability Testing - SAT 2009*. Ed. by Oliver Kullmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 73–79. ISBN: 978-3-642-02777-2 (cit. on p. 67).

[DG84] William F. Dowling and Jean H. Gallier. “Linear-time algorithms for testing the satisfiability of propositional horn formulae”. In: *The Journal of Logic Programming* 1.3 (1984), pp. 267–284. ISSN: 0743-1066. DOI: 10.1016/0743-1066(84)90014-1 (cit. on pp. 38, 67).

[DF13] Rodney G Downey and Michael Fellows. *Fundamentals of parameterized complexity*. en. Texts in computer science. London, England: Springer, 2013 (cit. on p. 67).

[Dub+93] Olivier Dubois, Pascal André, Yacine Boufkhad, et al. “SAT versus UNSAT”. In: *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993*. Ed. by David S. Johnson and Michael A. Trick. Vol. 26. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1993, pp. 415–436. DOI: 10.1090/dimacs/026/20 (cit. on p. 50).

[DD01] Olivier Dubois and Gilles Dequen. “A Backbone-Search Heuristic for Efficient Solving of Hard 3-SAT Formulae”. In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI’01. Seattle, WA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 248–253. ISBN: 1558608125 (cit. on pp. 51, 52, 67).

[Edm65] Jack Edmonds. “Paths, Trees, and Flowers”. In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467. DOI: 10.4153/CJM-1965-045-4 (cit. on p. 31).

[EB05] Niklas Eén and Armin Biere. “Effective Preprocessing in SAT Through Variable and Clause Elimination”. In: *Theory and Applications of Satisfiability Testing*. Ed. by Fahiem Bacchus and Toby Walsh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 61–75. ISBN: 978-3-540-31679-4 (cit. on p. 58).

[ES03a] Niklas Eén and Niklas Sörensson. “An Extensible SAT-solver”. In: *Theory and Applications of Satisfiability Testing*. Ed. by Enrico Giunchiglia and Armando Tacchella. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 502–518. ISBN: 978-3-540-24605-3 (cit. on pp. 42, 52, 56).

[ES03b] Niklas Eén and Niklas Sörensson. “Temporal Induction by Incremental SAT Solving”. In: *Electronic Notes in Theoretical Computer Science* 89.4 (2003). BMC’2003, First International Workshop on Bounded Model Checking, pp. 543–560. ISSN: 1571-0661. DOI: https://doi.org/10.1016/S1571-0661(05)82542-3 (cit. on p. 108).

[ES06a] Niklas Eén and Niklas Sörensson. “Translating Pseudo-Boolean Constraints into SAT”. In: *J. Satisf. Boolean Model. Comput.* 2.1-4 (2006), pp. 1–26. DOI: 10.3233/sat190014 (cit. on pp. 101, 102).

[ES06b] Niklas Eén and Niklas Sörensson. “Translating Pseudo-Boolean Constraints into SAT”. In: *J. Satisf. Boolean Model. Comput.* 2.1-4 (2006), pp. 1–26. DOI: 10.3233/sat190014 (cit. on p. 106).

[Eld11] William Palin Elderton. *Frequency curves and correlation*. Cambridge University Press, 2011 (cit. on p. 77).

[FLX16] Zhiwen Fang, Chu-Min Li, and Ke Xu. “An Exact Algorithm Based on MaxSAT Reasoning for the Maximum Weight Clique Problem”. In: *J. Artif. Intell. Res.* 55 (2016), pp. 799–833. DOI: 10.1613/jair.4953 (cit. on p. 85).

[Fen+17] Yu Feng, Osbert Bastani, Ruben Martins, et al. “Automated Synthesis of Semantic Malware Signatures using Maximum Satisfiability”. In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. URL: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/automated-synthesis-semantic-malware-signatures-using-maximum-satisfiability/ (cit. on p. 85).

[FT05] Valnir Ferreira and John Thornton. “Tie Breaking in Clause Weighting Local Search for SAT”. In: *AI 2005: Advances in Artificial Intelligence*. Ed. by Shichao Zhang and Ray Jarvis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 70–81. ISBN: 978-3-540-31652-7 (cit. on p. 65).

[Fil+20] Yuval Filmus, Meena Mahajan, Gaurav Sood, et al. “MaxSAT Resolution and Subcube Sums”. In: *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*. Ed. by Luca Pulina and Martina Seidl. Vol. 12178. Lecture Notes in Computer Science. Springer, 2020, pp. 295–311. DOI: 10.1007/978-3-030-51825-7_21 (cit. on pp. 20, 113, 115, 151).

[Fou+07] Olivier Fourdrinoy, Éric Grégoire, Bertrand Mazure, et al. “Eliminating Redundant Clauses in SAT Instances”. In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Ed. by Pascal Van Hentenryck and Laurence Wolsey. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 71–83. ISBN: 978-3-540-72397-4 (cit. on p. 58).

[Fre95] Jon William Freeman. “Improvements to propositional satisfiability search algorithms”. PhD thesis. University of Pennsylvania, 1995. URL: https://repository.upenn.edu/dissertations/AAI9532175 (cit. on pp. 50, 58, 87).

[FB+99] Ehud Friedgut, Jean Bourgain, et al. “Sharp thresholds of graph properties, and the k-sat problem”. In: *Journal of the American mathematical Society* 12.4 (1999), pp. 1017–1054 (cit. on p. 66).

[Fu+18] Huimin Fu, Yang Xu, Guanfeng Wu, et al. “An Improved Adaptive Genetic Algorithm for Solving 3-SAT Problems Based on Effective Restart and Greedy Strategy”. In: *Int. J. Comput. Intell. Syst.* 11.1 (2018), pp. 402–413. DOI: 10.2991/ijcis.11.1.30 (cit. on p. 66).

[Fu07] Zhaohui Fu. “Extending the power of Boolean satisfiability solvers: Techniques and applications”. PhD thesis. Princeton University, 2007 (cit. on p. 104).

[FM06a] Zhaohui Fu and Sharad Malik. “On Solving the Partial MAX-SAT Problem”. In: *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*. Ed. by Armin Biere and Carla P. Gomes. Vol. 4121. Lecture Notes in Computer Science. Springer, 2006, pp. 252–265. DOI: 10.1007/11814948_25 (cit. on p. 85).

[FM06b] Zhaohui Fu and Sharad Malik. “On Solving the Partial MAX-SAT Problem”. In: *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*. Ed. by Armin Biere and Carla P. Gomes. Vol. 4121. Lecture Notes in Computer Science. Springer, 2006, pp. 252–265. DOI: 10.1007/11814948_25 (cit. on pp. 101, 104).

[GH11] Oliver Gableske and Marijn J. H. Heule. “EagleUP: Solving Random 3-SAT Using SLS with Unit Propagation”. In: *Theory and Applications of Satisfiability Testing - SAT 2011*. Ed. by Karem A. Sakallah and Laurent Simon. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 367–368. ISBN: 978-3-642-21581-0 (cit. on p. 66).

[GS07] Matteo Gagliolo and Jürgen Schmidhuber. “Learning Restart Strategies”. In: *Proceedings of IJCAI*. 2007, pp. 792–797 (cit. on p. 80).

[GJ79] Michael R Garey and David S Johnson. *Computers and intractability*. New York, NY: W.H. Freeman, Apr. 1979 (cit. on p. 32).

[GC11] Aurélien Garivier and Olivier Cappé. “The KL-UCB algorithm for bounded stochastic bandits and beyond”. In: *Proceedings of the 24th annual conference on learning theory*. JMLR Workshop and Conference Proceedings. 2011, pp. 359–376 (cit. on p. 79).

[GK22] Serge Gaspers and Andrew Kaploun. “Faster Algorithms for Weak Backdoors”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.4 (2022), pp. 3741–3748. DOI: 10.1609/aaai.v36i4.20288 (cit. on p. 67).

[GS12a] Serge Gaspers and Stefan Szeider. “Backdoors to Acyclic SAT”. In: *Automata, Languages, and Programming*. Ed. by Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 363–374. ISBN: 978-3-642-31594-7 (cit. on p. 67).

[GS12b] Serge Gaspers and Stefan Szeider. “Backdoors to Satisfaction”. In: *The Multivariate Algorithmic Revolution and Beyond: Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*. Ed. by Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 287–317. ISBN: 978-3-642-30891-8. DOI: 10.1007/978-3-642-30891-8_15 (cit. on p. 67).

[Gel02] Allen Van Gelder. “Generalizations of Watched Literals for Backtracking Search”. In: *Annals of Mathematics and Artificial Intelligence* (2002) (cit. on p. 47).

[08] “Generalizing backdoors”. English. In: *Proceedings of the 5th International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS 2008)*. 5th International Workshop on Local Search Techniques in Constraint Satisfaction (LSCS 2008) ; Conference date: 14-09-2008. 2008 (cit. on p. 67).

[GW94a] Ian P Gent and Toby Walsh. “The SAT phase transition”. In: *ECAI*. Vol. 94. PITMAN. 1994, pp. 105–109 (cit. on p. 66).

[Gen13] Ian P. Gent. “Optimal Implementation of Watched Literals and More General Techniques”. In: 48.1 (2013), pp. 231–252. ISSN: 1076-9757 (cit. on p. 47).

[GW93] Ian P. Gent and Toby Walsh. “Towards an Understanding of Hill-Climbing Procedures for SAT”. In: *Proceedings of the Eleventh National Conference on Artificial Intelligence*. AAAI’93. Washington, D.C.: AAAI Press, 1993, pp. 28–33. ISBN: 0262510715 (cit. on pp. 62, 63).

[GWH95] Ian Philip Gent, T Walsh, and J Hallam. “Unsatisfied Variables in Local Search”. English. In: Hybrid Problems, Hybrid Solutions; Proc AISB-95. 1995, pp. 73–85 (cit. on pp. 62, 63).

[Gen64] Gerhard Gentzen. “Investigations into Logical Deduction”. In: *American Philosophical Quarterly* 1.4 (1964), pp. 288–306. ISSN: 00030481. URL: http://www.jstor.org/stable/20009142 (cit. on p. 70).

[GM19] Bishwamittra Ghosh and Kuldeep S. Meel. “IMLI: An Incremental Framework for MaxSAT-Based Learning of Interpretable Classification Rules”. In: *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society, AIES 2019, Honolulu, HI, USA, January 27-28, 2019*. Ed. by Vincent Conitzer, Gillian K. Hadfield, and Shannon Vallor. ACM, 2019, pp. 203–210. DOI: 10.1145/3306618.3314283 (cit. on p. 85).

[GSL19] Michael Gimelfarb, Scott Sanner, and Chi-Guhn Lee. “Epsilon-BMC: A Bayesian Ensemble Approach to Epsilon-Greedy Exploration in Model-Free Reinforcement Learning”. In: *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*. Ed. by Amir Globerson and Ricardo Silva. Vol. 115. Proceedings of Machine Learning Research. AUAI Press, 2019, pp. 476–485. URL: http://proceedings.mlr.press/v115/gimelfarb20a.html (cit. on p. 77).

[GL15] Jesús Giráldez-Cru and Jordi Levy. “A modularity-based random SAT instances generator”. In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015 (cit. on p. 68).

[GL16] Jesús Giráldez-Cru and Jordi Levy. “Generating SAT instances with community structure”. In: *Artificial Intelligence* 238 (2016), pp. 119–134. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2016.06.001 (cit. on p. 68).

[GL17] Jesús Giráldez-Cru and Jordi Levy. “Locality in Random SAT Instances”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 638–644. DOI: 10.24963/ijcai.2017/89 (cit. on p. 67).

[GM07] Enrico Giunchiglia and Marco Maratea. “Planning as Satisfiability with Preferences”. In: *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*. AAAI Press, 2007, pp. 987–992. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-157.php (cit. on p. 102).

[GM11] Enrico Giunchiglia and Marco Maratea. “Introducing Preferences in Planning as Satisfiability”. In: *J. Log. Comput.* 21.2 (2011), pp. 205–229. DOI: 10.1093/logcom/exq023 (cit. on p. 102).

[Glo89] Fred Glover. “Tabu Search—Part I”. In: *ORSA Journal on Computing* 1.3 (1989), pp. 190–206. DOI: 10.1287/ijoc.1.3.190 (cit. on p. 63).

[GW94b] Michel X. Goemans and David P. Williamson. “New 3/4-Approximation Algorithms for the Maximum Satisfiability Problem”. In: *SIAM J. Discret. Math.* 7.4 (1994), pp. 656–666. DOI: 10.1137/S0895480192243516 (cit. on p. 112).

[Goe91] Andreas Goerdt. “Cutting plane versus frege proof systems”. In: *Computer Science Logic*. Ed. by Egon Börger, Hans Kleine Büning, Michael M. Richter, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 174–194. ISBN: 978-3-540-38401-4 (cit. on p. 71).

[GN02] E Goldberg and BerkMin Novikov. “A Fast and Robust Sat-Solver, 2002 Design”. In: *Automation and Test in Europe Conference and Exhibition (DATE'02)*. 2002 (cit. on pp. 52, 53, 56, 57).

[GN07] Eugene Goldberg and Yakov Novikov. “BerkMin: A fast and robust Sat-solver”. In: *Discrete Applied Mathematics* 155.12 (2007). SAT 2001, the Fourth International Symposium on the Theory and Applications of Satisfiability Testing, pp. 1549–1561. ISSN: 0166-218X. DOI: https://doi.org/10.1016/j.dam.2006.10.007 (cit. on p. 53).

[Gol08] Oded Goldreich. *Computational complexity: A Conceptual Perspective*. Cambridge, England: Cambridge University Press, Apr. 2008 (cit. on p. 29).

[Gom+00] Carla P Gomes, Bart Selman, Nuno Crato, et al. “Heavy-tailed phenomena in satisfiability and constraint satisfaction problems”. In: *Journal of automated reasoning* 24.1-2 (2000), pp. 67–100. DOI: 10.1023/A:1006314320276 (cit. on pp. 56, 123).

[GSC97] Carla P. Gomes, Bart Selman, and Nuno Crato. “Heavy-tailed distributions in combinatorial search”. In: *Principles and Practice of Constraint Programming-CP97*. Ed. by Gert Smolka. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 121–135. ISBN: 978-3-540-69642-1 (cit. on p. 56).

[GSK98] Carla P. Gomes, Bart Selman, and Henry Kautz. “Boosting Combinatorial Search through Randomization”. In: *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*. AAAI '98/IAAI '98. Madison, Wisconsin, USA: American Association for Artificial Intelligence, 1998, pp. 431–437. ISBN: 0262510987 (cit. on pp. 56, 57).

[GV98] Jens Gottlieb and Nico Voss. “Improving the Performance of Evolutionary Algorithms for the Satisfiability Problem by Refining Functions”. In: *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*. PPSN V. Berlin, Heidelberg: Springer-Verlag, 1998, pp. 755–764. ISBN: 3540650784 (cit. on p. 66).

[Gra+10] Ana Graça, Inês Lynce, João Marques-Silva, et al. “Efficient and Accurate Haplotype Inference by Combining Parsimony and Pedigree Information”. In: *Algebraic and Numeric Biology - 4th International Conference, ANB 2010, Hagenberg, Austria, July 31- August 2, 2010, Revised Selected Papers*. Ed. by Katsuhisa Horimoto, Masahiko Nakatsui, and Nikolaj Popov. Vol. 6479. Lecture Notes in Computer Science. Springer, 2010, pp. 38–56. DOI: 10.1007/978-3-642-28067-2_3 (cit. on p. 85).

[GML11] Ana Graça, João Marques-Silva, and Inês Lynce. “Haplotype Inference Using Propositional Satisfiability”. In: *Mathematical Approaches to Polymer Sequence Analysis and Related Problems*. Ed. by Renato Bruni. Springer, 2011, pp. 127–147. DOI: 10.1007/978-1-4419-6800-5_7 (cit. on p. 85).

[GLM14] Éric Grégoire, Jean-Marie Lagniez, and Bertrand Mazure. “Multiple Contraction through Partial-Max-SAT”. In: *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*. IEEE Computer Society, 2014, pp. 321–327. DOI: 10.1109/ICTAI.2014.56 (cit. on p. 85).

[Gré+05] Éric Grégoire, Richard Ostrowski, Bertrand Mazure, et al. “Automatic Extraction of Functional Dependencies”. In: *Theory and Applications of Satisfiability Testing*. Ed. by Holger H. Hoos and David G. Mitchell. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 122–132. ISBN: 978-3-540-31580-3 (cit. on pp. 59, 67).

[Gu92] Jun Gu. “Efficient Local Search for Very Large-Scale Satisfiability Problems”. In: *SIGART Bull.* 3.1 (1992), pp. 8–12. ISSN: 0163-5719. URL: https://doi.org/10.1145/130836.130837 (cit. on p. 60).

[GL12] João Guerra and Inês Lynce. “Reasoning over Biological Networks Using Maximum Satisfiability”. In: *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*. Ed. by Michela Milano. Vol. 7514. Lecture Notes in Computer Science. Springer, 2012, pp. 941–956. DOI: 10.1007/978-3-642-33558-7_67 (cit. on p. 85).

[Guo+14] L. Guo, S. Jabbour, J. Lonlac, et al. “Diversification by Clauses Deletion Strategies in Portfolio Parallel SAT Solving”. In: *2014 IEEE 26th International Conference on Tools with Artificial Intelligence (ICTAI)*. Los Alamitos, CA, USA: IEEE Computer Society, 2014, pp. 701–708. DOI: 10.1109/ICTAI.2014.110 (cit. on p. 56).

[Guo+10] Long Guo, Youssef Hamadi, Said Jabbour, et al. “Diversification and Intensification in Parallel SAT Solving”. In: *Principles and Practice of Constraint Programming – CP 2010*. Ed. by David Cohen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 252–265. ISBN: 978-3-642-15396-9 (cit. on p. 59).

[GGW06] Aarti Gupta, Malay K. Ganai, and Chao Wang. “SAT-Based Verification Methods and Applications in Hardware Verification”. In: *Formal Methods for Hardware Verification, 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2006, Bertinoro, Italy, May 22-27, 2006, Advanced Lectures*. Ed. by Marco Bernardo and Alessandro Cimatti. Vol. 3965. Lecture Notes in Computer Science. Springer, 2006, pp. 108–143. DOI: 10.1007/11757283_5 (cit. on p. 39).

[Gup11] Arjun K. Gupta. “Beta Distribution”. In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 144–145. ISBN: 978-3-642-04898-2. DOI: 10.1007/978-3-642-04898-2_144 (cit. on p. 77).

[Hab+02] Djamal Habet, Chu Min Li, Laure Devendeville, et al. “A Hybrid Approach for SAT”. In: *Principles and Practice of Constraint Programming - CP 2002*. Ed. by Pascal Van Hentenryck. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 172–184. ISBN: 978-3-540-46135-7 (cit. on p. 59).

[HT21] Djamal Habet and Cyril Terrioux. “Conflict history based heuristic for constraint satisfaction problem solving”. In: *J. Heuristics* 27.6 (2021), pp. 951–990. DOI: 10.1007/s10732-021-09475-z (cit. on pp. 80, 120, 134).

[HTA13] Djamal Habet, Donia Toumi, and André Abramé. “Ncca+: Configuration checking and novelty+ like heuristic”. In: *Proceedings of SAT Competition: Solver and Benchmark Descriptions*. Department of Computer Science Report Series B B-2013-1 (2013). Ed. by Adrian Balint, Anton Belov, Marijn Heule, et al., p. 62 (cit. on p. 64).

[Häh01] Reiner Hähnle. “Chapter 3 - Tableaux and Related Methods”. In: *Handbook of Automated Reasoning*. Ed. by Alan Robinson and Andrei Voronkov. Handbook of Automated Reasoning. Amsterdam: North-Holland, 2001, pp. 101–178. ISBN: 978-0-444-50813-3. DOI: https://doi.org/10.1016/B978-044450813-3/50005-9 (cit. on p. 71).

[Hai09] Wang Hai-Xiao. “Dijkstra algorithm in the shortest path of logistics transporting”. In: *Journal of Value Engineering* (2009) (cit. on p. 25).

[HH14] Shai Haim and Marijn Heule. “Towards Ultra Rapid Restarts”. In: *CoRR* abs/1402.4413 (2014). URL: http://arxiv.org/abs/1402.4413 (cit. on p. 57).

[HJS12] Youssef Hamadi, Said Jabbour, and Jabbour Sais. “Control-Based Clause Sharing in Parallel SAT Solving”. In: *Autonomous Search*. Ed. by Youssef Hamadi, Eric Monfroy, and Frédéric Saubion. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 245–267. ISBN: 978-3-642-21434-9. DOI: 10.1007/978-3-642-21434-9_10 (cit. on p. 59).

[HJS10a] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. “ManySAT: a parallel SAT solver”. In: *Journal on Satisfiability, Boolean Modeling and Computation* 6.4 (2010), pp. 245–262 (cit. on p. 59).

[HJS10b] YOUSSEF HAMADI, SAÏD JABBOUR, and LAKHDAR SAÏS. “LEARNING FOR DYNAMIC SUBSUMPTION”. In: *International Journal on Artificial Intelligence Tools* 19.04 (2010), pp. 511–529 (cit. on p. 56).

[HS07] Hyojung Han and Fabio Somenzi. “Alembic: An Efficient Algorithm for CNF Preprocessing”. In: *Proceedings of the 44th Annual Design Automation Conference*. DAC ’07. San Diego, California: Association for Computing Machinery, 2007, pp. 582–587. ISBN: 9781595936271. DOI: 10.1145/1278480.1278628 (cit. on p. 58).

[HS09] Hyojung Han and Fabio Somenzi. “On-the-Fly Clause Improvement”. In: *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing*. SAT ’09. Swansea, UK: Springer-Verlag, 2009, pp. 209–222. ISBN: 9783642027765. DOI: 10.1007/978-3-642-02777-2_21 (cit. on p. 58).

[HJ90] Pierre Hansen and Brigitte Jaumard. “Algorithms for the Maximum Satisfiability Problem”. In: *Computing* 44.4 (Apr. 1990), pp. 279–303. ISSN: 0010-485X. DOI: 10.1007/BF02241270 (cit. on p. 111).

[HE80] Robert M. Haralick and Gordon L. Elliott. “Increasing tree search efficiency for constraint satisfaction problems”. In: *Artificial Intelligence* 14.3 (1980), pp. 263–313. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(80)90051-X (cit. on p. 52).

[HG95] William D Harvey and Matthew L Ginsberg. “Limited discrepancy search”. In: *IJCAI (1)*. 1995, pp. 607–615 (cit. on p. 123).

[Hås97] Johan Håstad. “Some Optimal Inapproximability Results”. In: *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*. STOC ’97. El Paso, Texas, USA: Association for Computing Machinery, 1997, pp. 1–10. ISBN: 0897918886. DOI: 10.1145/258533.258536 (cit. on p. 112).

[Hås01] Johan Håstad. “Some Optimal Inapproximability Results”. In: *J. ACM* 48.4 (July 2001), pp. 798–859. ISSN: 0004-5411. DOI: 10.1145/502090.502098 (cit. on p. 112).

[Has+09] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009 (cit. on p. 75).

[Hei13] "Begriffsschrift, a formula language, modeled upon that of arithmetic, for pure thought: GOTTLOB FREGE(1879)". In: *Two Fundamental Texts in Mathematical Logic*. Ed. by Jean van Heijenoort. Cambridge, MA and London, England: Harvard University Press, 2013, pp. 1–82. DOI: doi:10.4159/harvard.9780674864603.c2 (cit. on p. 70).

[HLO07] Federico Heras, Javier Larrosa, and Albert Oliveras. "MiniMaxSat: A New Weighted Max-SAT Solver". In: *Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings*. Ed. by João Marques-Silva and Karem A. Sakallah. Vol. 4501. Lecture Notes in Computer Science. Springer, 2007, pp. 41–55. DOI: 10.1007/978-3-540-72788-0_8 (cit. on pp. 86, 88).

[HLO08] Federico Heras, Javier Larrosa, and Albert Oliveras. "MiniMaxSAT: An efficient weighted Max-SAT solver". In: *Journal of Artificial Intelligence Research* 31 (2008), pp. 1–32 (cit. on pp. 19, 86, 88, 95, 99).

[HM11] Federico Heras and João Marques-Silva. "Read-Once Resolution for Unsatisfiability-Based Max-SAT Algorithms". In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. Ed. by Toby Walsh. IJCAI/AAAI, 2011, pp. 572–577. DOI: 10.5591/978-1-57735-516-8/IJCAI11-103 (cit. on pp. 106, 114, 115, 179, 181).

[HMM11] Federico Heras, António Morgado, and João Marques-Silva. "Core-Guided Binary Search Algorithms for Maximum Satisfiability". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. Ed. by Wolfram Burgard and Dan Roth. AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3713 (cit. on pp. 102, 105).

[HU09] Alexander Hertel and Alasdair Urquhart. "Algorithms and Complexity Results for Input and Unit Resolution". In: *Journal of Satisfiability, Boolean Modeling and Computation* 6 (2009). DOI: 10.3233/SAT190066 (cit. on p. 72).

[Her+08] Philipp Hertel, Fahiem Bacchus, Toniann Pitassi, et al. "Clause Learning Can Effectively P-Simulate General Propositional Resolution". In: *AAAI*. 2008, pp. 283–290 (cit. on p. 71).

[HJB10] Marijn Heule, Matti Järvisalo, and Armin Biere. "Clause Elimination Procedures for CNF Formulas". In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Ed. by Christian G. Fermüller and Andrei Voronkov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 357–371. ISBN: 978-3-642-16242-8 (cit. on p. 58).

[HJS19] Marijn Heule, Matti Järvisalo, and Martin Suda, eds. *Proceedings of SAT Race 2019: Solver and Benchmark Descriptions*. Vol. B-2019-1. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, 2019. URL: https://helda.helsinki.fi/bitstream/handle/10138/306988/sr2019_proceedings.pdf (cit. on p. 128).

[HJS+18] Marijn Heule, Matti Juhani Järvisalo, Martin Suda, et al., eds. *Proceedings of SAT Competition 2018: Solver and Benchmark Descriptions*. Vol. B-2018-1. Department of Computer Science Series of Publications B. Department of Computer Science, University of Helsinki, 2018. URL: https://helda.helsinki.fi/bitstream/handle/10138/237063/sc2018_proceedings.pdf (cit. on p. 128).

[HJW13] Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. "Trimming while checking clausal proofs". In: *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*. IEEE, 2013, pp. 181–188. URL: https://ieeexplore.ieee.org/document/6679408/ (cit. on p. 72).

[HHW13] Marijn J. H. Heule, Warren A. Hunt, and Nathan Wetzler. “Verifying Refutations with Extended Resolution”. In: *Automated Deduction – CADE-24*. Ed. by Maria Paola Bonacina. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 345–359. ISBN: 978-3-642-38574-2 (cit. on p. 72).

[HJB11] Marijn J. H. Heule, Matti Järvisalo, and Armin Biere. “Efficient CNF Simplification Based on Binary Implication Graphs”. In: *Theory and Applications of Satisfiability Testing - SAT 2011*. Ed. by Karem A. Sakallah and Laurent Simon. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 201–215. ISBN: 978-3-642-21581-0 (cit. on p. 59).

[Heu+12] Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, et al. “Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads”. In: *Hardware and Software: Verification and Testing*. Ed. by Kerstin Eder, João Lourenço, and Onn Shehory. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 50–65. ISBN: 978-3-642-34188-5 (cit. on p. 59).

[HB19] Randy Hickey and Fahiem Bacchus. “Speeding Up Assumption-Based SAT”. In: *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*. Ed. by Mikolás Janota and Inês Lynce. Vol. 11628. Lecture Notes in Computer Science. Springer, 2019, pp. 164–182. URL: https://doi.org/10.1007/978-3-030-24258-9%5C_11 (cit. on p. 109).

[HK05] Edward A. Hirsch and Arist Kojevnikov. “UnitWalk: A New SAT Solver That Uses Local Search Guided by Unit Clause Elimination”. In: *Annals of Mathematics and Artificial Intelligence* 43.1–4 (2005), pp. 91–111. ISSN: 1012-2443. DOI: 10.1007/s10472-004-9421-4 (cit. on p. 66).

[Hoa69] Charles Antony Richard Hoare. “An axiomatic basis for computer programming”. In: *Communications of the ACM* 12.10 (1969), pp. 576–580 (cit. on p. 26).

[HN13] Steffen Hölldobler and Van-Hau Nguyen. “An efficient encoding of the at-most-one constraint”. In: *Technical Report 1304. Technische Universitäat Dresden* (2013) (cit. on p. 100).

[Hoo99] Holger H. Hoos. “On the Run-Time Behaviour of Stochastic Local Search Algorithms for SAT”. In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*. AAAI ’99/IAAI ’99. Orlando, Florida, USA: American Association for Artificial Intelligence, 1999, pp. 661–666. ISBN: 0262511061 (cit. on pp. 62, 63).

[Hoo02] Holger H. Hoos. “An Adaptive Noise Mechanism for WalkSAT”. In: *Eighteenth National Conference on Artificial Intelligence*. Edmonton, Alberta, Canada: American Association for Artificial Intelligence, 2002, pp. 655–660. ISBN: 0262511290 (cit. on pp. 65, 66).

[HS00] Holger H. Hoos and Thomas Stützle. “Local Search Algorithms for SAT: An Empirical Evaluation”. In: *J. Autom. Reason.* 24.4 (2000), pp. 421–481. DOI: 10.1023/A:1006350622830 (cit. on p. 62).

[Hua07] Jinbo Huang. “The Effect of Restarts on the Efficiency of Clause Learning”. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI’07. Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pp. 2318–2323 (cit. on p. 57).

[HTH02] Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos. “Scaling and Probabilistic Smoothing: Efficient Dynamic Local Search for SAT”. In: *Principles and Practice of Constraint Programming - CP 2002*. Ed. by Pascal Van Hentenryck. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 233–248. ISBN: 978-3-540-46135-7 (cit. on pp. 65, 66, 111).

[Hyt+17] Antti Hyttinen, Sergey M. Plis, Matti Järvisalo, et al. “A constraint optimization approach to causal discovery from subsampled time series data”. In: *Int. J. Approx. Reason.* 90 (2017), pp. 208–225. DOI: 10.1016/j.ijar.2017.07.009 (cit. on p. 85).

[HSJ17] Antti Hyttinen, Paul Saikko, and Matti Järvisalo. “A Core-Guided Approach to Learning Optimal Causal Graphs”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. Ed. by Carles Sierra. ijcai.org, 2017, pp. 645–651. DOI: 10.24963/ijcai.2017/90 (cit. on p. 85).

[IMM17] Alexey Ignatiev, António Morgado, and João Marques-Silva. “On Tackling the Limits of Resolution in SAT Solving”. In: *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*. Ed. by Serge Gaspers and Toby Walsh. Vol. 10491. Lecture Notes in Computer Science. Springer, 2017, pp. 164–183. DOI: 10.1007/978-3-319-66263-3_11 (cit. on p. 113).

[IMM19] Alexey Ignatiev, António Morgado, and João Marques-Silva. “RC2: an Efficient MaxSAT Solver”. In: *J. Satisf. Boolean Model. Comput.* 11.1 (2019), pp. 53–64. DOI: 10.3233/SAT190116 (cit. on pp. 105, 106, 109).

[Ign+18] Alexey Ignatiev, Filipe Pereira, Nina Narodytska, et al. “A SAT-Based Approach to Learn Explainable Decision Sets”. In: *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*. Ed. by Didier Galmiche, Stephan Schulz, and Roberto Sebastiani. Vol. 10900. Lecture Notes in Computer Science. Springer, 2018, pp. 627–645. DOI: 10.1007/978-3-319-94205-6_41 (cit. on p. 85).

[Ign+15] Alexey Ignatiev, Alessandro Previti, Mark H. Liffiton, et al. “Smallest MUS Extraction with Minimal Hitting Set Dualization”. In: *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*. Ed. by Gilles Pesant. Vol. 9255. Lecture Notes in Computer Science. Springer, 2015, pp. 173–182. DOI: 10.1007/978-3-319-23219-5_13 (cit. on p. 110).

[IBJ21] Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. “Refined Core Relaxation for Core-Guided MaxSAT Solving”. In: *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*. Ed. by Laurent D. Michel. Vol. 210. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 28:1–28:19. DOI: 10.4230/LIPIcs.CP.2021.28 (cit. on p. 105).

[IBJ22] Hannes Ihalainen, Jeremias Berg, and Matti Järvisalo. “Clause Redundancy and Preprocessing in Maximum Satisfiability”. In: *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*. Ed. by Jasmin Blanchette, Laura Kovács, and Dirk Pattinson. Vol. 13385. Lecture Notes in Computer Science. Springer, 2022, pp. 75–94. DOI: 10.1007/978-3-031-10769-6_6 (cit. on p. 91).

[Ise20] Markus Iser. “Recognition and Exploitation of Gate Structure in SAT Solving”. PhD thesis. Karlsruhe Institute of Technology, Germany, 2020. URL: https://nbn-resolving.org/urn:nbn:de:101:1-202004290459566073264 8 (cit. on pp. 59, 68).

[IMS15] Markus Iser, Norbert Manthey, and Carsten Sinz. “Recognition of Nested Gates in CNF Formulas”. In: *Theory and Applications of Satisfiability Testing – SAT 2015*. Ed. by Marijn Heule and Sean Weaver. Cham: Springer International Publishing, 2015, pp. 255–271. ISBN: 978-3-319-24318-4 (cit. on p. 68).

[IM95] Kazuo Iwama and Eiji Miyano. “Intractability of Read-Once Resolution”. In: *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*. IEEE Computer Society, 1995, pp. 29–36. DOI: 10.1109/SCT.1995.514725 (cit. on pp. 72, 73).

[Jab+14] Saïd Jabbour, Jerry Lonlac, Lakhdar Sais, et al. “Revisiting the Learned Clauses Database Reduction Strategies”. In: *CoRR* abs/1402.1956 (2014). URL: http://arxiv.org/abs/1402.1956 (cit. on p. 56).

[JCZ16] Gerold Jäger, Sharlee Climer, and Weixiong Zhang. “The complete parsimony haplotype inference problem and algorithms based on integer programming, branch-and-bound and Boolean satisfiability”. In: *J. Discrete Algorithms* 37 (2016), pp. 68–83. DOI: 10.1016/j.jda.2016.06.001 (cit. on p. 85).

[JM17] Sima Jamali and David Mitchell. “Improving SAT Solver Performance with Structure-based Preferential Bumping”. In: *GCAI 2017, 3rd Global Conference on Artificial Intelligence, Miami, FL, USA, 18-22 October 2017*. Ed. by Christoph Benzmüller, Christine L. Lisetti, and Martin Theobald. Vol. 50. EPiC Series in Computing. EasyChair, 2017, pp. 175–187. DOI: 10.29007/3pxg (cit. on p. 69).

[JM18] Sima Jamali and David Mitchell. “Centrality-Based Improvements to CDCL Heuristics”. In: *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*. Ed. by Olaf Beyersdorff and Christoph M. Wintersteiger. Vol. 10929. Lecture Notes in Computer Science. Springer, 2018, pp. 122–131. DOI: 10.1007/978-3-319-94144-8_8 (cit. on p. 69).

[JLM15] Mikolás Janota, Inês Lynce, and João Marques-Silva. “Algorithms for computing backbones of propositional formulae”. In: *AI Commun.* 28.2 (2015), pp. 161–177. DOI: 10.3233/AIC-140640 (cit. on p. 67).

[JBH10] Matti Järvisalo, Armin Biere, and Marijn Heule. “Blocked Clause Elimination”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Javier Esparza and Rupak Majumdar. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 129–144. ISBN: 978-3-642-12002-2 (cit. on p. 58).

[Jär+12] Matti Järvisalo, Daniel Le Berre, Olivier Roussel, et al. “The international SAT solver competitions”. In: *Ai Magazine* 33.1 (2012), pp. 89–92 (cit. on p. 39).

[JN08] Matti Järvisalo and Ilkka Niemelä. “The effect of structural branching on the efficiency of clause learning SAT solving: An experimental study”. In: *J. Algorithms* 63.1-3 (2008), pp. 90–113. DOI: 10.1016/j.jalgor.2008.02.005 (cit. on p. 55).

[JW90] Robert G. Jeroslow and Jinchang Wang. “Solving propositional satisfiability problems”. In: *Annals of Mathematics and Artificial Intelligence* 1.1-4 (1990), pp. 167–187. DOI: 10.1007/bf01531077 (cit. on pp. 49, 87).

[Jia+18] Hua Jiang, Chu-Min Li, Yanli Liu, et al. “A Two-Stage MaxSAT Reasoning Approach for the Maximum Weight Clique Problem”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 1338–1346. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16809 (cit. on p. 85).

[JKS95] Yuejun Jiang, Henry Kautz, and Bart Selman. “Solving problems with hard and soft constraints using a stochastic algorithm for MAX-SAT”. In: *1st International Joint Workshop on Artificial Intelligence and Operations Research*. Vol. 20. 1995 (cit. on p. 85).

[JS05] HoonSang Jin and Fabio Somenzi. “An Incremental Algorithm to Check Satisfiability for Bounded Model Checking”. In: *Electronic Notes in Theoretical Computer Science* 119.2 (2005). Proceedings of the 2nd International Workshop on Bounded Model Checking (BMC 2004), pp. 51–65. ISSN: 1571-0661. DOI: https://doi.org/10.1016/j.entcs.2004.06.062 (cit. on p. 58).

[Joh73] David S. Johnson. “Approximation Algorithms for Combinatorial Problems”. In: *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*. Ed. by Alfred V. Aho, Allan Borodin, Robert L. Constable, et al. ACM, 1973, pp. 38–49. DOI: 10.1145/800125.804034 (cit. on p. 112).

[Joh74] David S. Johnson. “Approximation algorithms for combinatorial problems”. In: *Journal of Computer and System Sciences* 9.3 (1974), pp. 256–278. ISSN: 0022-0000. DOI: https://doi.org/10.1016/S0022-0000(74)80044-9 (cit. on p. 112).

[JM11] Manu Jose and Rupak Majumdar. “Cause clue clauses: error localization using maximum satisfiability”. In: *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*. Ed. by Mary W. Hall and David A. Padua. ACM, 2011, pp. 437–446. DOI: 10.1145/1993498.1993550 (cit. on p. 85).

[Jos+18] Saurabh Joshi, Prateek Kumar, Ruben Martins, et al. “Approximation Strategies for Incomplete MaxSAT”. In: *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*. Ed. by John N. Hooker. Vol. 11008. Lecture Notes in Computer Science. Springer, 2018, pp. 219–228. URL: https://doi.org/10.1007/978-3-319-98334-9%5C_15 (cit. on p. 112).

[Jos+19] Saurabh Joshi, Prateek Kumar, Sukrut Rao, et al. “Open-WBO-Inc: Approximation Strategies for Incomplete Weighted MaxSAT”. In: *J. Satisf. Boolean Model. Comput.* 11.1 (2019), pp. 73–97. DOI: 10.3233/SAT190118 (cit. on p. 112).

[Juk12] Stasys Jukna. “Cutting Plane Proofs”. In: *Boolean Function Complexity: Advances and Frontiers*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 521–556. DOI: 10.1007/978-3-642-24508-4_19 (cit. on p. 70).

[Kad+10] Serdar Kadioglu, Yuri Malitsky, Meinolf Sellmann, et al. “ISAC - Instance-Specific Algorithm Configuration”. In: *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*. Ed. by Helder Coelho, Rudi Studer, and Michael J. Wooldridge. Vol. 215. Frontiers in Artificial Intelligence and Applications. IOS Press, 2010, pp. 751–756. DOI: 10.3233/978-1-60750-606-5-751 (cit. on p. 111).

[Kah11] Daniel Kahneman. *Thinking, Fast and Slow*. New York, NY: Farrar, Straus & Giroux, Oct. 2011 (cit. on p. 18).

[KZ97] Howard Karloff and Uri Zwick. “A 7/8-Approximation Algorithm for MAX 3SAT?” In: *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*. FOCS ’97. USA: IEEE Computer Society, 1997, p. 406. ISBN: 0818681977 (cit. on p. 112).

[Kar72] Richard M. Karp. “Reducibility Among Combinatorial Problems”. In: *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*. Ed. by Raymond E. Miller and James W. Thatcher. The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9 (cit. on pp. 32, 38).

[KS12] George Katsirelos and Laurent Simon. “Eigenvector Centrality in Industrial SAT Instances”. In: *Principles and Practice of Constraint Programming*. Ed. by Michela Milano. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 348–356. ISBN: 978-3-642-33558-7 (cit. on p. 69).

[KCG12] Emilie Kaufmann, Olivier Cappe, and Aurelien Garivier. “On Bayesian Upper Confidence Bounds for Bandit Problems”. In: *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Neil D. Lawrence and Mark Girolami. Vol. 22. Proceedings of Machine Learning Research. La Palma, Canary Islands: PMLR, 2012, pp. 592–600. URL: https://proceedings.mlr.press/v22/kaufmann12.html (cit. on p. 79).

[KR18] M.M. Khwārizmī and F. Rosen. *The Algebra of Mohammed Ben Musa. Edited and Translated by Frederic Rosen*. Creative Media Partners, LLC, 2018. ISBN: 9781376742800. URL: https://books.google.fr/books?id=4u4StAEACAAJ (cit. on p. 26).

[Kie+18] Benjamin Kiesl, Martina Seidl, Hans Tompits, et al. “Local Redundancy in SAT: Generalizations of Blocked Clauses”. In: *Log. Methods Comput. Sci.* 14.4 (2018). DOI: 10.23638/LMCS-14(4:3)2018 (cit. on p. 58).

[Kil+05] Philip Kilby, John Slaney, Sylvie Thiébaux, et al. “Backbones and Backdoors in Satisfiability”. In: *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*. AAAI'05. Pittsburgh, Pennsylvania: AAAI Press, 2005, pp. 1368–1373. ISBN: 157735236x (cit. on pp. 51, 67).

[Kim+17] Chang Kyu Kim, Sung Won Kim, Huy Hung Nguyen, et al. “Path planning for automatic guided vehicle with multiple target points in known environment”. In: *International Conference on Advanced Engineering Theory and Applications*. Springer. 2017, pp. 726–735 (cit. on p. 24).

[Kle36] S. C. Kleene. “λ-Definability and Recursiveness”. In: *Duke Mathematical Journal* 2.2 (1936), pp. 340–353. DOI: 10.1215/S0012-7094-36-00227-2 (cit. on p. 27).

[Kle37] S. C. Kleene. “General Recursive Functions of Natural Numbers”. In: *Journal of Symbolic Logic* 2.1 (1937), pp. 38–38. DOI: 10.2307/2268801 (cit. on p. 27).

[Kle52] Stephen Cole Kleene. “Introduction to metamathematics”. In: (1952) (cit. on p. 28).

[KT05] Jon Kleinberg and Eva Tardos. *Algorithm Design*. en. Upper Saddle River, NJ: Pearson, Mar. 2005 (cit. on pp. 29, 30).

[KK84] William Kneale and Martha Kneale. *The development of logic*. en. London, England: Oxford University Press, Nov. 1984 (cit. on p. 33).

[Kor96] Richard E Korf. “Improved limited discrepancy search”. In: *AAAI/IAAI, Vol. 1*. 1996, pp. 286–291.

[Kor+17] Tuukka Korhonen, Jeremias Berg, Paul Saikko, et al. “MaxPre: An Extended MaxSAT Preprocessor”. In: *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*. Ed. by Serge Gaspers and Toby Walsh. Vol. 10491. Lecture Notes in Computer Science. Springer, 2017, pp. 449–456. DOI: 10.1007/978-3-319-66263-3_28 (cit. on p. 91).

[Kor+22] Frédéric Koriche, Christophe Lecoutre, Anastasia Paparrizou, et al. “Best Heuristic Identification for Constraint Satisfaction”. In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*. Ed. by Luc De Raedt. ijcai.org, 2022, pp. 1859–1865. DOI: 10.24963/ijcai.2022/258 (cit. on pp. 80, 120, 122, 132).

[Kos+12] Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, et al. “QMaxSAT: A Partial Max-SAT Solver”. In: *J. Satisf. Boolean Model. Comput.* 8.1/2 (2012), pp. 95–100. DOI: 10.3233/sat190091 (cit. on pp. 101, 102).

[KKS08] Stephan Kottler, Michael Kaufmann, and Carsten Sinz. “Computation of Renameable Horn Backdoors”. In: *Theory and Applications of Satisfiability Testing – SAT 2008*. Ed. by Hans Kleine Büning and Xishun Zhao. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 154–160. ISBN: 978-3-540-79719-7 (cit. on p. 67).

[Kro21] Daniel Kroening. “Software Verification”. In: *Handbook of Satisfiability - Second Edition*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, et al. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 791–818. DOI: 10.3233/FAIA201004 (cit. on p. 39).

[Kro67] M. R. Krom. “The Decision Problem for a Class of First-Order Formulas in Which all Disjunctions are Binary”. In: *Mathematical Logic Quarterly* 13.1-2 (1967), pp. 15–20. DOI: 10.1002/malq.19670130104 (cit. on pp. 38, 67).

[Kue12] Adrian Kuegel. “Improved Exact Solver for the Weighted MAX-SAT Problem”. In: *POS-10. Pragmatics of SAT*. Ed. by Daniel Le Berre. Vol. 8. EPiC Series in Computing. EasyChair, 2012, pp. 15–27. DOI: 10.29007/38lm (cit. on p. 150).

[Küe12] Adrian Küegel. “Improved Exact Solver for the Weighted MAX-SAT Problem”. In: *POS-10. Pragmatics of SAT*. Ed. by Daniel Le Berre. Vol. 8. EPiC Series in Computing. EasyChair, 2012, pp. 15–27 (cit. on pp. 86, 114).

[Kül12] Adrian Kügel. “Natural Max-SAT Encoding of Min-SAT”. In: *Learning and Intelligent Optimization*. Ed. by Youssef Hamadi and Marc Schoenauer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 431–436. ISBN: 978-3-642-34413-8 (cit. on p. 85).

[Kul99] O. Kullmann. “On a generalization of extended resolution”. In: *Discrete Applied Mathematics* 96-97 (1999), pp. 149–176. ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(99)00037-2 (cit. on p. 58).

[Kun86] Sukhamay Kundu. “Tree resolution and generalized semantic tree”. In: *Proceedings of the ACM SIGART International Symposium on Methodologies for Intelligent Systems, ISMIS 1986, Knoxville, Tennessee, USA, October 22-24, 1986*. Ed. by Zbigniew W. Ras and Maria Zemankova. ACM, 1986, pp. 270–278. DOI: 10.1145/12808.12838 (cit. on pp. 72, 73).

[Kur+19] Vitaly Kurin, Saad Godil, Shimon Whiteson, et al. “Improving SAT Solver Heuristics with Graph Networks and Reinforcement Learning. CoRR, abs/1909.11830”. In: *arXiv preprint arXiv:1909.11830* (2019) (cit. on p. 80).

[Kur+20] Vitaly Kurin, Saad Godil, Shimon Whiteson, et al. “Can Q-Learning with Graph Networks Learn a Generalizable Branching Heuristic for a SAT Solver?” In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, et al. Vol. 33. Curran Associates, Inc., 2020, pp. 9608–9621. URL: https://proceedings.neurips.cc/paper/2020/file/6d70cb65d15211726dcce4c0e971e21c-Paper.pdf (cit. on p. 80).

[LJH14] Frédéric Lafitte, Jorge Nakahara Jr., and Dirk Van Heule. “Applications of SAT Solvers in Cryptanalysis: Finding Weak Keys and Preimages”. In: *J. Satisf. Boolean Model. Comput.* 9.1 (2014), pp. 1–25. DOI: 10.3233/sat190099 (cit. on p. 39).

[LR85] Tze Leung Lai and Herbert Robbins. “Asymptotically efficient adaptive allocation rules”. In: *Advances in applied mathematics* 6.1 (1985), pp. 4–22. DOI: 10.1016/0196-8858(85)90002-8 (cit. on pp. 75, 79).

[LD60] A. H. Land and A. G. Doig. “An Automatic Method of Solving Discrete Programming Problems”. In: *Econometrica* 28.3 (1960), pp. 497–520. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1910129 (cit. on p. 85).

[LT92] Tracy Larrabee and Yumi Tsuji. *EVIDENCE FOR A SATISFIABILITY THRESHOLD FOR RANDOM 3CNF FORMULAS*. Tech. rep. USA, 1992 (cit. on p. 66).

[LH05a] Javier Larrosa and Federico Heras. “Resolution in Max-SAT and its relation to local consistency in weighted CSPs”. In: *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*. Ed. by Leslie Pack Kaelbling and Alessandro Saffiotti. Professional Book Center, 2005, pp. 193–198. URL: http://ijcai.org/Proceedings/05/Papers/0360.pdf (cit. on pp. 89, 90, 113, 114, 116, 151, 169).

[LHG08] Javier Larrosa, Federico Heras, and Simon de Givry. “A logical approach to efficient Max-SAT solving”. In: *Artif. Intell.* 172.2-3 (2008), pp. 204–233. DOI: 10.1016/j.artint.2007.05.006 (cit. on pp. 20, 89–91, 95, 113–116, 151, 152, 161, 162, 169).

[LR20a] Javier Larrosa and Emma Rollon. “Augmenting the Power of (Partial) MaxSat Resolution with Extension”. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 1561–1568. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5516 (cit. on pp. 20, 113, 116, 151).

[LR20b] Javier Larrosa and Emma Rollon. “Towards a Better Understanding of (Partial Weighted) MaxSAT Proof Systems”. In: *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*. Ed. by Luca Pulina and Martina Seidl. Vol. 12178. Lecture Notes in Computer Science. Springer, 2020, pp. 218–232. DOI: 10.1007/978-3-030-51825-7_16 (cit. on pp. 20, 113, 115, 116, 151, 153, 159, 176).

[LS20] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020 (cit. on pp. 75, 76).

[Laz+12] Nadjib Lazaar, Youssef Hamadi, Said Jabbour, et al. *Cooperation control in Parallel SAT Solving: a Multi-armed Bandit Approach*. Research Report RR-8070. INRIA, 2012, p. 18 (cit. on pp. 59, 80).

[Le 01] Daniel Le Berre. "Exploiting the real power of unit propagation lookahead". In: *Electronic Notes in Discrete Mathematics* 9 (2001). LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001), pp. 59–80. ISSN: 1571-0653. DOI: https://doi.org/10.1016/S1571-0653(04)00314-2 (cit. on p. 58).

[Le +17] Ludovic Le Frioux, Souheib Baarir, Julien Sopena, et al. "PaInleSS: A Framework for Parallel SAT Solving". In: *Theory and Applications of Satisfiability Testing – SAT 2017*. Ed. by Serge Gaspers and Toby Walsh. Cham: Springer International Publishing, 2017, pp. 233–250. ISBN: 978-3-319-66263-3 (cit. on p. 59).

[Le +19] Ludovic Le Frioux, Souheib Baarir, Julien Sopena, et al. "Modular and Efficient Divide-and-Conquer SAT Solver on Top of the Painless Framework". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Tomáš Vojnar and Lijun Zhang. Cham: Springer International Publishing, 2019, pp. 135–151. ISBN: 978-3-030-17462-0 (cit. on p. 59).

[LC18] Zhendong Lei and Shaowei Cai. "Solving (Weighted) Partial MaxSAT by Dynamic Local Search for SAT". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. Ed. by Jérôme Lang. ijcai.org, 2018, pp. 1346–1352. DOI: 10.24963/ijcai.2018/187 (cit. on pp. 111, 112).

[LC20] Zhendong Lei and Shaowei Cai. "NuDist: An Efficient Local Search Algorithm for (Weighted) Partial MaxSAT". In: *Comput. J.* 63.9 (2020), pp. 1321–1337. DOI: 10.1093/comjnl/bxz063 (cit. on p. 111).

[LM08] Florian Letombe and Joao Marques-Silva. "Improvements to Hybrid Incremental SAT Algorithms". In: *Theory and Applications of Satisfiability Testing – SAT 2008*. Ed. by Hans Kleine Büning and Xishun Zhao. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 168–181. ISBN: 978-3-540-79719-7 (cit. on p. 59).

[LP97] Harry Lewis and Christos H Papadimitriou. *Elements of the theory of computation*. 2nd ed. Upper Saddle River, NJ: Pearson, Aug. 1997 (cit. on p. 27).

[Lew78] Harry R. Lewis. "Renaming a Set of Clauses as a Horn Set". In: *J. ACM* 25.1 (1978), pp. 134–135. ISSN: 0004-5411. DOI: 10.1145/322047.322059 (cit. on pp. 38, 67).

[LSB05] Matthew D. T. Lewis, Tobias Schubert, and Bernd W. Becker. "Speedup Techniques Utilized in Modern SAT Solvers". In: *Theory and Applications of Satisfiability Testing*. Ed. by Fahiem Bacchus and Toby Walsh. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 437–443. ISBN: 978-3-540-31679-4 (cit. on p. 57).

[Li99] Chu Min Li. "A constraint-based approach to narrow search trees for satisfiability". In: *Information Processing Letters* 71.2 (1999), pp. 75–80. ISSN: 0020-0190. DOI: https://doi.org/10.1016/S0020-0190(99)00088-5 (cit. on p. 52).

[LA97a] Chu Min Li and Anbulagan. "Look-ahead versus look-back for satisfiability problems". In: *Principles and Practice of Constraint Programming-CP97*. Ed. by Gert Smolka. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 341–355. ISBN: 978-3-540-69642-1 (cit. on p. 46).

[LA97b] Chu Min Li and Anbulagan Anbulagan. "Heuristics Based on Unit Propagation for Satisfiability Problems". In: *Proceedings of the 15th International Joint Conference on Artifical Intelligence - Volume 1*. IJCAI'97. Nagoya, Japan: Morgan Kaufmann Publishers Inc., 1997, pp. 366–371 (cit. on pp. 50, 52).

[LH05b] Chu Min Li and Wen Qi Huang. “Diversification and Determinism in Local Search for Satisfiability”. In: SAT’05. St Andrews, UK: Springer-Verlag, 2005, pp. 158–172. ISBN: 3540262768. DOI: 10.1007/11499107_12 (cit. on pp. 62, 63).

[LL12] Chu Min Li and Yu Li. “Satisfying versus Falsifying in Local Search for Satisfiability - (Poster Presentation)”. In: *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*. Ed. by Alessandro Cimatti and Roberto Sebastiani. Vol. 7317. Lecture Notes in Computer Science. Springer, 2012, pp. 477–478. DOI: 10.1007/978-3-642-31612-8_43 (cit. on pp. 62, 66).

[LM21] Chu Min Li and Felip Manyà. “MaxSAT, Hard and Soft Constraints”. In: *Handbook of Satisfiability - Second Edition*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, et al. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 903–927. DOI: 10.3233/FAIA201007 (cit. on pp. 19, 85).

[LM22] Chu Min Li and Felip Manyà. “Inference in MaxSAT and MinSAT”. In: *The Logic of Software. A Tasting Menu of Formal Methods - Essays Dedicated to Reiner Hähnle on the Occasion of His 60th Birthday*. Ed. by Wolfgang Ahrendt, Bernhard Beckert, Richard Bubel, et al. Vol. 13360. Lecture Notes in Computer Science. Springer, 2022, pp. 350–369. DOI: 10.1007/978-3-031-08166-8_16 (cit. on p. 181).

[Li+08] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, et al. “Transforming Inconsistent Subformulas in MaxSAT Lower Bound Computation”. In: *Principles and Practice of Constraint Programming*. Ed. by Peter J. Stuckey. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 582–587. ISBN: 978-3-540-85958-1 (cit. on p. 86).

[Li+09] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, et al. “Exploiting Cycle Structures in Max-SAT”. In: *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*. Ed. by Oliver Kullmann. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009, pp. 467–480. DOI: 10.1007/978-3-642-02777-2_43 (cit. on p. 86).

[Li+10a] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, et al. “Resolution-based lower bounds in MaxSAT”. In: *Constraints* 15 (Oct. 2010), pp. 456–484 (cit. on pp. 19, 86, 88, 99, 144, 180).

[LMP05] Chu Min Li, Felip Manyà, and Jordi Planes. “Exploiting Unit Propagation to Compute Lower Bounds in Branch and Bound Max-SAT Solvers”. In: *Principles and Practice of Constraint Programming - CP 2005*. Ed. by Peter van Beek. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 403–414 (cit. on pp. 87, 89, 92, 135).

[LMP07] Chu Min Li, Felip Manyà, and Jordi Planes. “New Inference Rules for Max-SAT”. In: *Journal of Artificial Intelligence Research* 30 (Oct. 2007), pp. 321–359. ISSN: 1076-9757 (cit. on pp. 19, 86, 88, 90, 95, 99, 114, 135, 180).

[Li+10b] Chu Min Li, Felip Manyà, Zhe Quan, et al. “Exact MinSAT Solving”. In: *Theory and Applications of Satisfiability Testing – SAT 2010*. Ed. by Ofer Strichman and Stefan Szeider. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 363–368. ISBN: 978-3-642-14186-7 (cit. on p. 85).

[LMS16] Chu Min Li, Felip Manyà, and Joan Ramon Soler. “A Clause Tableau Calculus for MaxSAT”. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. Ed. by Subbarao Kambhampati. IJCAI/AAAI Press, 2016, pp. 766–772. URL: http://www.ijcai.org/Abstract/16/114 (cit. on p. 113).

[LQ10] Chu Min Li and Zhe Quan. “An Efficient Branch-and-Bound Algorithm Based on MaxSAT for the Maximum Clique Problem”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. Ed. by Maria Fox and David Poole. AAAI Press, 2010. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1611 (cit. on p. 85).

[LWL12] Chu Min Li, Wanxia Wei, and Yu Li. “Exploiting Historical Relationships of Clauses and Variables in Local Search for Satisfiability”. In: *Theory and Applications of Satisfiability Testing – SAT 2012*. Ed. by Alessandro Cimatti and Roberto Sebastiani. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 479–480. ISBN: 978-3-642-31612-8 (cit. on p. 62).

[LMP06] 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-06)*. Vol. 1. event-place: Boston, Massachusetts. Boston, Massachusetts: AAAI Press, 2006, pp. 86–91. ISBN: 978-1-57735-281-5 (cit. on p. 150).

[Li+20a] Chu-Min Li, Fan Xiao, Mao Luo, et al. “Clause Vivification by Unit Propagation in CDCL SAT Solvers”. In: *Artif. Intell.* 279.C (2020). ISSN: 0004-3702. DOI: 10.1016/j.artint.2019.103197 (cit. on p. 58).

[Li+21a] Chu-Min Li, Zhenxing Xu, Jordi Coll, et al. “Combining Clause Learning and Branch and Bound for MaxSAT”. In: *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*. Ed. by Laurent D. Michel. Vol. 210. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 38:1–38:18. DOI: 10.4230/LIPIcs.CP.2021.38 (cit. on pp. 86, 88, 91, 134, 150, 181).

[Li+22a] Chu-Min Li, Zhenxing Xu, Jordi Coll, et al. “Boosting branch-and-bound MaxSAT solvers with clause learning”. In: *AI Commun.* 35.2 (2022), pp. 131–151. DOI: 10.3233/AIC-210178 (cit. on pp. 86, 88, 91, 134, 150, 181).

[Li+21b] Chunxiao Li, Jonathan Chung, Soham Mukherjee, et al. “On the Hierarchical Community Structure of Practical Boolean Formulas”. In: *Theory and Applications of Satisfiability Testing – SAT 2021*. Ed. by Chu-Min Li and Felip Manyà. Cham: Springer International Publishing, 2021, pp. 359–376. ISBN: 978-3-030-80223-3 (cit. on p. 68).

[Li+20b] Chunxiao Li, Noah Fleming, Marc Vinyals, et al. “Towards a Complexity-Theoretic Understanding of Restarts in SAT Solvers”. In: *Theory and Applications of Satisfiability Testing – SAT 2020*. Ed. by Luca Pulina and Martina Seidl. Cham: Springer International Publishing, 2020, pp. 233–249. ISBN: 978-3-030-51825-7 (cit. on p. 57).

[LSB04] Xiao Yu Li, Matthias F. Stallmann, and Franc Brglez. “A Local Search SAT Solver Using an Effective Switching Strategy and an Efficient Unit Propagation”. In: *Theory and Applications of Satisfiability Testing*. Ed. by Enrico Giunchiglia and Armando Tacchella. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 53–68. ISBN: 978-3-540-24605-3 (cit. on p. 66).

[Li+22b] Yang Li, Yuqi Jia, Wanqian Luo, et al. “Kissat Adaptive Restart, Kissat Cfexp: Adaptive Restart Policy and Variable Scoring Improvement”. In: *SAT COMPETITION 2022*. Department of Computer Science Series of Publications B B-2022-1 (2022). Ed. by Tomas Balyo and Marijn Heule and Markus Iser and Matti Järvisalo and Martin Suda, p. 39 (cit. on p. 134).

[Li+20c] Yih-Lang Li, Shih-Ting Lin, Shinichi Nishizawa, et al. “MCell: Multi-Row Cell Layout Synthesis with Resource Constrained MAX-SAT Based Detailed Routing”. In: *Proceedings of the 39th International Conference on Computer-Aided Design*. ICCAD ’20. Virtual Event, USA: Association for Computing Machinery, 2020. ISBN: 9781450380263. DOI: 10.1145/3400302.3415612 (cit. on p. 85).

[LB11] Zijie Li and Peter van Beek. “Finding Small Backdoors in SAT Instances”. In: *Advances in Artificial Intelligence*. Ed. by Cory Butz and Pawan Lingras. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 269–280. ISBN: 978-3-642-21043-3 (cit. on p. 67).

[Lia+18a] Jia Liang, Hari Govind, Pascal Poupart, et al. “An Empirical Study of Branching Heuristics through the Lens of Global Learning Rate”. In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 5319–5323. DOI: 10.24963/ijcai.2018/745 (cit. on p. 55).

[Lia+17] Jia Hui Liang, Chanseok, Vijay Ganesh, et al. “MapleCOMSPS, MapleCOMSPS_LRB, MapleCOMSPS_CHB”. In: *Proceedings of SAT Competition 2017: Solver and Benchmark Descriptions*. Ed. by Tomáš Balyo, Marijn Heule, and Matti Järvisalo. 2017, pp. 20–21 (cit. on pp. 120, 121).

[Lia+16a] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, et al. “Exponential Recency Weighted Average Branching Heuristic for SAT Solvers”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. Ed. by Dale Schuurmans and Michael P. Wellman. AAAI Press, 2016, pp. 3434–3440. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12451 (cit. on pp. 19, 42, 52, 54, 57, 80, 120, 122, 132).

[Lia+16b] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, et al. “Learning Rate Based Branching Heuristic for SAT Solvers”. In: *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. Ed. by Nadia Creignou and Daniel Le Berre. Vol. 9710. Lecture Notes in Computer Science. Springer, 2016, pp. 123–140. DOI: 10.1007/978-3-319-40970-2_9 (cit. on pp. 42, 54, 57, 80, 120).

[Lia+15] Jia Hui Liang, Vijay Ganesh, Ed Zulkoski, et al. “Understanding VSIDS Branching Heuristics in Conflict-Driven Clause-Learning SAT Solvers”. In: *Hardware and Software: Verification and Testing*. Ed. by Nir Piterman. Cham: Springer International Publishing, 2015, pp. 225–241. ISBN: 978-3-319-26287-1 (cit. on p. 55).

[Lia+16c] Jia Hui Liang, Chanseok Oh, Vijay Ganesh, et al. “MapleCOMSPS, MapleCOMSPS_LRB, MapleCOMSPS_ CHB”. In: *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*. Ed. by Tomas Balyo, Marijn Heule, and Matti Järvisalo. Vol. B-2016-1. Department of Computer Science Report Series B. University of Helsinki, 2016, pp. 52–53 (cit. on pp. 42, 52, 54, 57, 88, 120, 121).

[Lia+18b] Jia Hui Liang, Chanseok Oh, Minu Mathew, et al. “Machine Learning-Based Restart Policy for CDCL SAT Solvers”. In: *Theory and Applications of Satisfiability Testing – SAT 2018*. Ed. by Olaf Beyersdorff and Christoph M. Wintersteiger. Cham: Springer International Publishing, 2018, pp. 94–110. ISBN: 978-3-319-94144-8 (cit. on p. 57).

[LZK16] Xiaojuan Liao, Hui Zhang, and Miyuki Koshimura. “Reconstructing AES Key Schedule Images with SAT and MaxSAT”. In: *IEICE Trans. Inf. Syst.* 99-D.1 (2016), pp. 141–150. DOI: 10.1587/transinf.2015EDP7223 (cit. on p. 85).

[Lia+19] Xiaojuan Liao, Hui Zhang, Miyuki Koshimura, et al. “Maximum Satisfiability Formulation for Optimal Scheduling in Overloaded Real-Time Systems”. In: *PRICAI 2019: Trends in Artificial Intelligence - 16th Pacific Rim International Conference on Artificial Intelligence, Cuvu, Yanuca Island, Fiji, August 26-30, 2019, Proceedings, Part I*. Ed. by Abhaya C. Nayak and Alok Sharma. Vol. 11670. Lecture Notes in Computer Science. Springer, 2019, pp. 618–631. DOI: 10.1007/978-3-030-29908-8_49 (cit. on p. 85).

[Lia+21] Xiaojuan Liao, Hui Zhang, Miyuki Koshimura, et al. “Modeling and Solving Scheduling in Overloaded Situations with Weighted Partial MaxSAT”. In: *Mathematical Problems in Engineering* 2021 (2021) (cit. on p. 85).

[Lib00] Paolo Liberatore. “On the complexity of choosing the branching literal in DPLL”. In: *Artificial Intelligence* 116.1 (2000), pp. 315–326. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(99)00097-1 (cit. on p. 52).

[Lit+63] John DC Little, Katta G Murty, Dura W Sweeney, et al. “An algorithm for the traveling salesman problem”. In: *Operations research* 11.6 (1963), pp. 972–989 (cit. on p. 85).

[LPR22] Daniel Lokshtanov, Fahad Panolan, and M. S. Ramanujan. “Backdoor Sets on Nowhere Dense SAT”. In: *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Ed. by Mikołaj Bojańczyk, Emanuela Merelli, and David P. Woodruff. Vol. 229. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 91:1–91:20. ISBN: 978-3-95977-235-8. DOI: 10.4230/LIPIcs.ICALP.2022.91 (cit. on p. 67).

[LN17] Jerry Lonlac and Engelbert Mephu Nguifo. “Towards Learned Clauses Database Reduction Strategies Based on Dominance Relationship”. In: *CoRR* abs/1705.10898 (2017). URL: http://arxiv.org/abs/1705.10898 (cit. on p. 56).

[Lov70] D. W. Loveland. “A linear format for resolution”. In: *Symposium on Automatic Demonstration*. Ed. by M. Laudet, D. Lacombe, L. Nolin, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 1970, pp. 147–162. ISBN: 978-3-540-36262-3. DOI: 10.1007/BFb0060630 (cit. on pp. 72, 74).

[Lov15] Jana Lovíšková. “Solving the 3-SAT problem using genetic algorithms”. In: *2015 IEEE 19th International Conference on Intelligent Engineering Systems (INES)*. IEEE. 2015, pp. 207–212 (cit. on p. 66).

[LH12] Zhipeng Lü and Jin-Kao Hao. “Adaptive memory-based local search for MAX-SAT”. In: *Applied Soft Computing* 12.8 (2012), pp. 2063–2071. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2012.01.013 (cit. on p. 111).

[LSZ93] Michael Luby, Alistair Sinclair, and David Zuckerman. “Optimal speedup of Las Vegas algorithms”. In: *Information Processing Letters* (1993), pp. 128–133. DOI: 10.1109/ISTCS.1993.253477 (cit. on p. 57).

[Luo+17a] Chuan Luo, Shaowei Cai, Kaile Su, et al. “CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability”. In: *Artif. Intell.* 243 (2017), pp. 26–44. DOI: 10.1016/j.artint.2016.11.001 (cit. on p. 111).

[Luo+14] Chuan Luo, Shaowei Cai, Wei Wu, et al. “Double Configuration Checking in Stochastic Local Search for Satisfiability”. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*. Ed. by Carla E. Brodley and Peter Stone. AAAI Press, 2014, pp. 2703–2709. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8174 (cit. on pp. 62, 64).

[Luo+15] Chuan Luo, Shaowei Cai, Wei Wu, et al. “CCLS: An Efficient Local Search Algorithm for Weighted Maximum Satisfiability”. In: *IEEE Trans. Computers* 64.7 (2015), pp. 1830–1843. DOI: 10.1109/TC.2014.2346196 (cit. on pp. 111, 112).

[LSC12] Chuan Luo, Kaile Su, and Shaowei Cai. “Improving Local Search for Random 3-SAT Using Quantitative Configuration Checking”. In: *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*. Ed. by Luc De Raedt, Christian Bessiere, Didier Dubois, et al. Vol. 242. Frontiers in Artificial Intelligence and Applications. IOS Press, 2012, pp. 570–575. DOI: 10.3233/978-1-61499-098-7-570 (cit. on pp. 62, 64).

[Luo+17b] Mao Luo, Chu-Min Li, Fan Xiao, et al. “An Effective Learnt Clause Minimization Approach for CDCL SAT Solvers”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017, pp. 703–711. DOI: 10.24963/ijcai.2017/98 (cit. on p. 56).

[LM04] I. Lynce and J. P. Marques-Silva. “Hidden Structure in Unsatisfiable Random 3-SAT: an Empirical Study”. In: *Proceedings of the International Conference on Tools with Artificial Intelligence (01/11/04)*. Event Dates: November 2004. 2004. URL: https://eprints.soton.ac.uk/262054/ (cit. on p. 67).

[LMM18] Inês Lynce, Vasco Manquinho, and Ruben Martins. “Parallel Maximum Satisfiability”. In: *Handbook of Parallel Constraint Reasoning*. Ed. by Youssef Hamadi and Lakhdar Sais. Cham: Springer International Publishing, 2018, pp. 61–99. ISBN: 978-3-319-63516-3. DOI: 10.1007/978-3-319-63516-3_3 (cit. on p. 111).

[LM05] Inês Lynce and João Marques-Silva. “Efficient Data Structures for Backtrack Search SAT Solvers”. In: *Annals of Mathematics and Artificial Intelligence* 43.1–4 (2005), pp. 137–152. ISSN: 1012-2443. DOI: 10.1007/s10472-004-9425-0 (cit. on pp. 47, 48).

[LM06] Inês Lynce and João Marques-Silva. “SAT in Bioinformatics: Making the Case with Haplotype Inference”. In: *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*. Ed. by Armin Biere and Carla P. Gomes. Vol. 4121. Lecture Notes in Computer Science. Springer, 2006, pp. 136–141. DOI: 10.1007/11814948_16 (cit. on p. 39).

[MNH08] Hans van Maaren, Linda van Norden, and M. J. H. Heule. “Sums of squares based approximation algorithms for MAX-SAT”. In: *Discret. Appl. Math.* 156.10 (2008), pp. 1754–1779. DOI: 10.1016/j.dam.2007.08.036 (cit. on p. 112).

[MFM05] Yogesh S. Mahajan, Zhaohui Fu, and Sharad Malik. “Zchaff2004: An Efficient SAT Solver”. In: *Theory and Applications of Satisfiability Testing*. Ed. by Holger H. Hoos and David G. Mitchell. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 360–375. ISBN: 978-3-540-31580-3 (cit. on p. 42).

[MSV21] Nikolas Mählmann, Sebastian Siebertz, and Alexandre Vigny. “Recursive Backdoors for SAT”. In: *46th International Symposium on Mathematical Foundations of Computer Science, MFCS 2021, August 23-27, 2021, Tallinn, Estonia*. Ed. by Filippo Bonchi and Simon J. Puglisi. Vol. 202. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 73:1–73:18. DOI: 10.4230/LIPIcs.MFCS.2021.73 (cit. on p. 67).

[MM18] Dmitry Malioutov and Kuldeep S. Meel. “MLIC: A MaxSAT-Based Framework for Learning Interpretable Classification Rules”. In: *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*. Ed. by John N. Hooker. Vol. 11008. Lecture Notes in Computer Science. Springer, 2018, pp. 312–327. DOI: 10.1007/978-3-319-98334-9_21 (cit. on p. 85).

[Mal+13] Yuri Malitsky, Ashish Sabharwal, Horst Samulowitz, et al. “Algorithm Portfolios Based on Cost-Sensitive Hierarchical Clustering”. In: *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. Ed. by Francesca Rossi. IJCAI/AAAI, 2013, pp. 608–614. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6946 (cit. on p. 111).

[MVN12] Hratch Mangassarian, Andreas G. Veneris, and Farid N. Najm. “Maximum Circuit Activity Estimation Using Pseudo-Boolean Satisfiability”. In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 31.2 (2012), pp. 271–284. DOI: 10.1109/TCAD.2011.2169259 (cit. on p. 85).

[MOV07] Panagiotis Manolios, Marc Galceran Oms, and Sergi Oliva Valls. “Checking Pedigree Consistency with PCS”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Orna Grumberg and Michael Huth. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 339–342. ISBN: 978-3-540-71209-1 (cit. on p. 39).

[MSP09] Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. “Algorithms for Weighted Boolean Optimization”. In: *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*. Ed. by Oliver Kullmann. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009, pp. 495–508. DOI: 10.1007/978-3-642-02777-2_45 (cit. on p. 105).

[Man12] Norbert Manthey. “Coprocessor 2.0 – A Flexible CNF Simplifier”. In: *Theory and Applications of Satisfiability Testing – SAT 2012*. Ed. by Alessandro Cimatti and Roberto Sebastiani. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 436–441 (cit. on p. 64).

[Mar95] Joao Marques-Silva. “Search algorithms for satisfiability problems in combinational switching circuits”. PhD thesis. University of Michigan, 1995 (cit. on pp. 42, 44, 47, 50, 56).

[MJL10] Joao Marques-Silva, Mikoláš Janota, and Inês Lynce. “On Computing Backbones of Propositional Theories”. In: *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*. NLD: IOS Press, 2010, pp. 15–20. ISBN: 9781607506058 (cit. on p. 67).

[MLM09] Joao Marques-Silva, Inês Lynce, and Sharad Malik. “Conflict-Driven Clause Learning SAT Solvers”. In: *Handbook of Satisfiability*. 2009 (cit. on pp. 19, 40, 42, 71).

[Mar+11] João Marques-Silva, Josep Argelich, Ana Graça, et al. “Boolean lexicographic optimization: algorithms & applications”. In: *Ann. Math. Artif. Intell.* 62.3-4 (2011), pp. 317–343. DOI: 10.1007/s10472-011-9233-2 (cit. on p. 108).

[MM08] João Marques-Silva and Vasco M. Manquinho. “Towards More Effective Unsatisfiability-Based Maximum Satisfiability Algorithms”. In: *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*. Ed. by Hans Kleine Büning and Xishun Zhao. Vol. 4996. Lecture Notes in Computer Science. Springer, 2008, pp. 225–230. DOI: 10.1007/978-3-540-79719-7_21 (cit. on p. 105).

[MP07] João Marques-Silva and Jordi Planes. “On Using Unsatisfiability for Solving Maximum Satisfiability”. In: *CoRR* abs/0712.1097 (2007). URL: http://arxiv.org/abs/0712.1097 (cit. on p. 105).

[MP08] João Marques-Silva and Jordi Planes. “Algorithms for Maximum Satisfiability using Unsatisfiable Cores”. In: *Design, Automation and Test in Europe, DATE 2008, Munich, Germany, March 10-14, 2008*. Ed. by Donatella Sciuto. ACM, 2008, pp. 408–413. DOI: 10.1109/DATE.2008.4484715 (cit. on pp. 102, 105).

[MS00] João P. Marques-Silva and Karem A. Sakallah. “Boolean Satisfiability in Electronic Design Automation”. In: *Proceedings of the 37th Annual Design Automation Conference*. DAC ’00. Los Angeles, California, USA: Association for Computing Machinery, 2000, pp. 675–680. ISBN: 1581131879. DOI: 10.1145/337292.337611 (cit. on p. 39).

[Mar+14a] Ruben Martins, Saurabh Joshi, Vasco Manquinho, et al. “Incremental Cardinality Constraints for MaxSAT”. In: *Principles and Practice of Constraint Programming*. Ed. by Barry O’Sullivan. Cham: Springer International Publishing, 2014, pp. 531–548. ISBN: 978-3-319-10428-7 (cit. on p. 106).

[Mar+14b] Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, et al. “Incremental Cardinality Constraints for MaxSAT”. In: *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*. Ed. by Barry O’Sullivan. Vol. 8656. Lecture Notes in Computer Science. Springer, 2014, pp. 531–548. DOI: 10.1007/978-3-319-10428-7_39 (cit. on p. 109).

[MML14] Ruben Martins, Vasco Manquinho, and Inês Lynce. “Open-WBO: A Modular MaxSAT Solver,” in: *Theory and Applications of Satisfiability Testing – SAT 2014*. Ed. by Carsten Sinz and Uwe Egly. Cham: Springer International Publishing, 2014, pp. 438–445. ISBN: 978-3-319-09284-3 (cit. on p. 105).

[Mas16] Kseniia Masliakova. “Optimal routing and charging procedures for electric buses”. MA thesis. UiT Norges arktiske universitet, 2016 (cit. on p. 24).

[Mas20] Jake Massimo. “An Analysis of Primality Testing and Its Use in Cryptographic Applications”. English. PhD thesis. Royal Holloway, University of London, 2020 (cit. on p. 24).

[Mat11] Robert Mateescu. *Treewidth in Industrial SAT Benchmarks*. Tech. rep. MSR-TR-2011-22. 2011. URL: https://www.microsoft.com/en-us/research/publication/treewidth-in-industrial-sat-benchmarks/ (cit. on p. 68).

[Mat+08] Paulo J. Matos, Jordi Planes, Florian Letombe, et al. “A MAX-SAT Algorithm Portfolio”. In: *ECAI 2008 - 18th European Conference on Artificial Intelligence, Patras, Greece, July 21-25, 2008, Proceedings*. Ed. by Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, et al. Vol. 178. Frontiers in Artificial Intelligence and Applications. IOS Press, 2008, pp. 911–912. DOI: 10.3233/978-1-58603-891-5-911 (cit. on p. 111).

[MSG97] Bertrand Mazure, Lakhdar Sais, and Éric Grégoire. “Tabu search for SAT”. In: *AAAI/IAAI*. 1997, pp. 281–285 (cit. on pp. 62, 63).

[MSG98] Bertrand Mazure, Lakhdar Sais, and Éric Grégoire. “Boosting Complete Techniques Thanks to Local Search Methods”. In: *Ann. Math. Artif. Intell.* 22.3-4 (1998), pp. 319–331. DOI: 10.1023/A:1018999721141 (cit. on p. 59).

[MSK97] David McAllester, Bart Selman, and Henry Kautz. “Evidence for Invariants in Local Search”. In: AAAI’97/IAAI’97. Providence, Rhode Island: AAAI Press, 1997, pp. 321–326. ISBN: 0262510952 (cit. on pp. 62, 63, 65).

[MP21] Paul Medvedev and Mihai Pop. “What do Eulerian and Hamiltonian cycles have to do with genome assembly?” In: *PLOS Computational Biology* 17.5 (May 2021), pp. 1–5. DOI: 10.1371/journal.pcbi.1008928 (cit. on p. 24).

[MDH10] Sid Mijnders, Boris De Wilde, and Marijn Heule. “Symbiosis of search and heuristics for random 3-SAT”. In: *Proceedings of the Third International Workshop on Logic and Search (LaSh 2010)*. Ed. by David Mitchell and Eugenia Ternovska. 2010 (cit. on p. 51).

[MSL92] David Mitchell, Bart Selman, and Hector Levesque. “Hard and Easy Distributions of SAT Problems”. In: *Proceedings of the Tenth National Conference on Artificial Intelligence*. AAAI’92. San Jose, California: AAAI Press, 1992, pp. 459–465. ISBN: 0262510634 (cit. on p. 66).

[MM97] Tom M Mitchell and Tom M Mitchell. *Machine learning*. Vol. 1. 9. McGraw-hill New York, 1997 (cit. on p. 75).

[Mon+99] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, et al. “Determining computational complexity from characteristic ‘phase transitions’”. In: *Nature* 400.6740 (1999), pp. 133–137. DOI: 10.1038/22055 (cit. on pp. 51, 67).

[MDM14] António Morgado, Carmine Dodaro, and João Marques-Silva. “Core-Guided MaxSAT with Soft Cardinality Constraints”. In: *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*. Ed. by Barry O’Sullivan. Vol. 8656. Lecture Notes in Computer Science. Springer, 2014, pp. 564–573. DOI: 10.1007/978-3-319-10428-7_41 (cit. on pp. 105, 106).

[Mor+13] António Morgado, Federico Heras, Mark H. Liffiton, et al. “Iterative and core-guided MaxSAT solving: A survey and assessment”. In: *Constraints An Int. J.* 18.4 (2013), pp. 478–534. DOI: 10.1007/s10601-013-9146-2 (cit. on pp. 100, 104).

[MHM12] António Morgado, Federico Heras, and João Marques-Silva. “Improvements to Core-Guided Binary Search for MaxSAT”. In: *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*. Ed. by Alessandro Cimatti and Roberto Sebastiani. Vol. 7317. Lecture Notes in Computer Science. Springer, 2012, pp. 284–297. DOI: 10.1007/978-3-642-31612-8_22 (cit. on p. 102).

[MIM14] António Morgado, Alexey Ignatiev, and João Marques-Silva. “MSCG: Robust Core-Guided MaxSAT Solving”. In: *J. Satisf. Boolean Model. Comput.* 9.1 (2014), pp. 129–134. DOI: 10.3233/sat190105 (cit. on pp. 105, 106, 109).

[Mor93] Paul Morris. “The Breakout Method for Escaping from Local Minima”. In: AAAI’93. Washington, D.C.: AAAI Press, 1993, pp. 40–45. ISBN: 0262510715 (cit. on p. 65).

[Mos+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, et al. “Chaff: Engineering an Efficient SAT Solver”. In: DAC ’01. Las Vegas, Nevada, USA: Association for Computing Machinery, 2001, pp. 530–535. ISBN: 1581132972. DOI: 10.1145/378239.379017 (cit. on pp. 19, 42, 47, 52, 56, 57, 119, 132).

[MFS16] Nathan Mull, Daniel J. Fremont, and Sanjit A. Seshia. “On the Hardness of SAT with Community Structure”. In: *Theory and Applications of Satisfiability Testing – SAT 2016*. Ed. by Nadia Creignou and Daniel Le Berre. Cham: Springer International Publishing, 2016, pp. 141–159. ISBN: 978-3-319-40970-2 (cit. on p. 68).

[Nad02] Alexander Nadel. “Backtrack search algorithms for propositional logic satisfiability: Review and innovations”. MA thesis. Hebrew University of Jerusalem, 2002 (cit. on p. 47).

[NB14] Nina Narodytska and Fahiem Bacchus. “Maximum Satisfiability Using Core-Guided MaxSAT Resolution”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 28.1 (June 2014). DOI: 10.1609/aaai.v28i1.9124 (cit. on pp. 106, 107, 114, 150, 181).

[Nej+17] Saeed Nejati, Zack Newsham, Joseph Scott, et al. “A Propagation Rate Based Splitting Heuristic for Divide-and-Conquer Solvers”. In: *Theory and Applications of Satisfiability Testing – SAT 2017*. Ed. by Serge Gaspers and Toby Walsh. Cham: Springer International Publishing, 2017, pp. 251–260. ISBN: 978-3-319-66263-3 (cit. on p. 59).

[New+14] Zack Newsham, Vijay Ganesh, Sebastian Fischmeister, et al. “Impact of Community Structure on SAT Solver Performance”. In: *Theory and Applications of Satisfiability Testing – SAT 2014*. Ed. by Carsten Sinz and Uwe Egly. Cham: Springer International Publishing, 2014, pp. 252–268. ISBN: 978-3-319-09284-3 (cit. on p. 68).

[New+15] Zack Newsham, William Lindsay, Vijay Ganesh, et al. “SATGraf: Visualizing the Evolution of SAT Formula Structure in Solvers”. In: *Theory and Applications of Satisfiability Testing – SAT 2015*. Ed. by Marijn Heule and Sean Weaver. Cham: Springer International Publishing, 2015, pp. 62–70 (cit. on pp. 68, 69).

[Ngh+08] Duc Nghia Pham, John Thornton, Charles Gretton, et al. “Combining adaptive and dynamic local search for satisfiability”. In: *Journal on Satisfiability, Boolean Modeling and Computation* 4.2-4 (2008), pp. 149–172 (cit. on pp. 62, 63, 66).

[Ngu+20] Van-Hau Nguyen, Van-Quyet Nguyen, Kyungbaek Kim, et al. “Empirical Study on SAT-Encodings of the At-Most-One Constraint”. In: *SMA 2020: The 9th International Conference on Smart Media and Applications, Jeju, Republic of Korea, September 17 - 19, 2020*. ACM, 2020, pp. 470–475. DOI: 10.1145/3426020.3426170 (cit. on p. 100).

[NR00] Rolf Niedermeier and Peter Rossmanith. “New Upper Bounds for Maximum Satisfiability”. In: *J. Algorithms* 36.1 (2000), pp. 63–88. DOI: 10.1006/jagm.2000.1075 (cit. on pp. 89, 95).

[NRS04] Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. “Detecting Backdoor Sets with Respect to Horn and Binary Clauses”. In: *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*. 2004. URL: http://www.satisfiability.org/SAT04/programme/51.pdf (cit. on p. 67).

[NRS06] Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. “Solving #SAT Using Vertex Covers”. In: *Theory and Applications of Satisfiability Testing - SAT 2006*. Ed. by Armin Biere and Carla P. Gomes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 396–409. ISBN: 978-3-540-37207-3 (cit. on p. 67).

[Nud+04] Eugene Nudelman, Kevin Leyton-Brown, Holger H. Hoos, et al. “Understanding Random SAT: Beyond the Clauses-to-Variables Ratio”. In: *Principles and Practice of Constraint Programming – CP 2004*. Ed. by Mark Wallace. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 438–452 (cit. on p. 111).

[Oh15] Chanseok Oh. “Between SAT and UNSAT: The Fundamental Difference in CDCL SAT”. In: *Theory and Applications of Satisfiability Testing – SAT 2015*. Ed. by Marijn Heule and Sean Weaver. Cham: Springer International Publishing, 2015, pp. 307–323. ISBN: 978-3-319-24318-4 (cit. on p. 57).

[Oh16] Chanseok Oh. “Improving SAT solvers by exploiting empirical characteristics of CDCL”. PhD thesis. New York University, 2016 (cit. on p. 56).

[OJ09] Emilia Oikarinen and Matti Järvisalo. “Max-ASP: Maximum satisfiability of answer set programs”. In: *International conference on logic programming and nonmonotonic reasoning*. Springer. 2009, pp. 236–249 (cit. on p. 111).

[Ost+06] R. Ostrowski, L. Sais, P. Siegel, et al. “Computing Horn Strong Backdoor Sets Thanks to Local Search”. In: *2006 18th IEEE International Conference on Tools with Artificial Intelligence*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 139–143. DOI: 10.1109/ICTAI.2006.43 (cit. on p. 67).

[Ost+02] Richard Ostrowski, Éric Grégoire, Bertrand Mazure, et al. “Recovering and Exploiting Structural Knowledge from CNF Formulas”. In: *Principles and Practice of Constraint Programming - CP 2002*. Ed. by Pascal Van Hentenryck. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 185–199. ISBN: 978-3-540-46135-7 (cit. on pp. 59, 67).

[OP09] Richard Ostrowski and Lionel Paris. “Detecting Boolean Functions for Proving Unsatisfiability”. In: *ICTAI 2009, 21st IEEE International Conference on Tools with Artificial Intelligence, Newark, New Jersey, USA, 2-4 November 2009*. IEEE Computer Society, 2009, pp. 409–416. DOI: 10.1109/ICTAI.2009.85 (cit. on pp. 59, 67).

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Sept. 1994. ISBN: 9780201530827 (cit. on pp. 29, 83).

[PW20] Anastasia Paparrizou and Hugues Wattez. “Perturbing Branching Heuristics in Constraint Solving”. In: *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*. Ed. by Helmut Simonis. Vol. 12333. Lecture Notes in Computer Science. Springer, 2020, pp. 496–513. DOI: 10.1007/978-3-030-58475-7_29 (cit. on pp. 80, 120–122, 132).

[PRB21] Tobias Paxian, Pascal Raiola, and Bernd Becker. “On Preprocessing for Weighted MaxSAT”. In: *Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings*. Ed. by Fritz Henglein, Sharon Shoham, and Yakir Vizel. Vol. 12597. Lecture Notes in Computer Science. Springer, 2021, pp. 556–577. DOI: 10.1007/978-3-030-67067-2_25.

[PRB18] Tobias Paxian, Sven Reimer, and Bernd Becker. “Pacose: An iterative SAT-based MaxSAT solver”. In: *MaxSAT Evaluation 2018: Solver and Benchmark Descriptions*. Ed. by F Bacchus, M J Järvisalo, and R Martins. Vol. B-2018-2. Department of Computer Science Report Series B. University of Helsinki, 2018, p. 20 (cit. on p. 102).

[Pie+21] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, et al. *Software Foundations*. Ed. by Benjamin C. Pierce. Vol. 1-2. Version 6.1. Electronic textbook, 2021. URL: https://softwarefoundations.cis.upenn.edu (cit. on p. 26).

[PHS08] Cédric Piette, Youssef Hamadi, and Lakhdar Saïs. “Vivifying Propositional Clausal Formulae”. In: *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence*. NLD: IOS Press, 2008, pp. 525–529. ISBN: 9781586038915 (cit. on p. 58).

[PD07a] Knot Pipatsrisawat and Adnan Darwiche. “A Lightweight Component Caching Scheme for Satisfiability Solvers”. In: *Theory and Applications of Satisfiability Testing – SAT 2007*. Ed. by João Marques-Silva and Karem A. Sakallah. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 294–299. ISBN: 978-3-540-72788-0 (cit. on pp. 48, 57).

[PD07b] Knot Pipatsrisawat and Adnan Darwiche. *RSat 2.0: SAT Solver Description*. Tech. rep. D–153. Automated Reasoning Group, Computer Science Department, UCLA, 2007 (cit. on p. 42).

[PD11] Knot Pipatsrisawat and Adnan Darwiche. "On the power of clause-learning SAT solvers as resolution engines". In: *Artificial Intelligence* 175.2 (2011), pp. 512–525. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2010.10.002 (cit. on p. 71).

[PS10] Toniann Pitassi and Rahul Santhanam. "Effectively Polynomial Simulations". In: *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*. Ed. by Andrew Chi-Chih Yao. Tsinghua University Press, 2010, pp. 370–382. URL: http://conference.iiis.tsinghua.edu.cn/ICS2010/content/papers/29.html (cit. on p. 71).

[PG86] David A. Plaisted and Steven Greenbaum. "A Structure-Preserving Clause Form Translation". In: *J. Symb. Comput.* 2.3 (Sept. 1986), pp. 293–304. ISSN: 0747-7171. DOI: 10.1016/S0747-7171(86)80028-1 (cit. on pp. 35, 68).

[Pla03] Jordi Planes. "Improved Branch and Bound Algorithms for Max-2-SAT and Weighted Max-2-SAT". In: *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings*. Ed. by Francesca Rossi. Vol. 2833. Lecture Notes in Computer Science. Springer, 2003, p. 991. DOI: 10.1007/978-3-540-45193-8_115 (cit. on p. 87).

[PW16] Matthias Poloczek and David P. Williamson. "An Experimental Evaluation of Fast Approximation Algorithms for the Maximum Satisfiability Problem". In: *Experimental Algorithms - 15th International Symposium, SEA 2016, St. Petersburg, Russia, June 5-8, 2016, Proceedings*. Ed. by Andrew V. Goldberg and Alexander S. Kulikov. Vol. 9685. Lecture Notes in Computer Science. Springer, 2016, pp. 246–261. DOI: 10.1007/978-3-319-38851-9_17 (cit. on p. 112).

[PW17] Matthias Poloczek and David P. Williamson. "An Experimental Evaluation of Fast Approximation Algorithms for the Maximum Satisfiability Problem". In: *ACM J. Exp. Algorithmics* 22 (2017). DOI: 10.1145/3064174 (cit. on p. 112).

[PWZ14] Matthias Poloczek, David P. Williamson, and Anke van Zuylen. "On Some Recent Approximation Algorithms for MAX SAT". In: *LATIN 2014: Theoretical Informatics - 11th Latin American Symposium, Montevideo, Uruguay, March 31 - April 4, 2014. Proceedings*. Ed. by Alberto Pardo and Alfredo Viola. Vol. 8392. Lecture Notes in Computer Science. Springer, 2014, pp. 598–609. DOI: 10.1007/978-3-642-54423-1_52 (cit. on p. 112).

[Pra65] Dag Prawitz. "Natural deduction: a proof-theoretical study". PhD thesis. Almqvist & Wiksell, 1965 (cit. on p. 70).

[Pre93] Daniele Pretolani. "Efficiency and stability of hypergraph SAT algorithms". In: *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993*. Ed. by David S. Johnson and Michael A. Trick. Vol. 26. DIMACS Series in Discrete Mathematics and Theoretical Computer Science. DIMACS/AMS, 1993, pp. 479–498. DOI: 10.1090/dimacs/026/23 (cit. on p. 50).

[PSM21] Nicolas Prevot, Mate Soos, and Kuldeep S. Meel. "Leveraging GPUs for Effective Clause Sharing in Parallel SAT Solving". In: *Theory and Applications of Satisfiability Testing – SAT 2021*. Ed. by Chu-Min Li and Felip Manyà. Cham: Springer International Publishing, 2021, pp. 471–487. ISBN: 978-3-030-80223-3 (cit. on p. 59).

[Pun07] Abraham P. Punnen. "The Traveling Salesman Problem: Applications, Formulations and Variations". In: *The Traveling Salesman Problem and Its Variations*. Ed. by Gregory Gutin and Abraham P. Punnen. Boston, MA: Springer US, 2007, pp. 1–28. ISBN: 978-0-306-48213-7. DOI: 10.1007/0-306-48213-4_1 (cit. on p. 26).

[Py21] Matthieu Py. "Inférence et certificats pour le problème de satisfiabilité maximum". PhD thesis. Aix-Marseille University, 2021. URL: https://www.theses.fr/2021AIXM0631 (cit. on pp. 20, 153, 173, 175, 177, 178).

[PCH20] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Towards Bridging the Gap Between SAT and Max-SAT Refutations”. In: *32nd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2020, Baltimore, MD, USA, November 9-11, 2020*. IEEE, 2020, pp. 137–144. DOI: 10.1109/ICTAI50040.2020.00032 (cit. on pp. 4, 20, 152).

[PCH21a] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “A Proof Builder for Max-SAT”. In: *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*. Ed. by Chu-Min Li and Felip Manyà. Vol. 12831. Lecture Notes in Computer Science. Springer, 2021, pp. 488–498. DOI: 10.1007/978-3-030-80223-3_33 (cit. on pp. 4, 20, 152, 178).

[PCH21b] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Computing Max-SAT Refutations using SAT Oracles”. In: *33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021*. IEEE, 2021, pp. 404–411. DOI: 10.1109/ICTAI52525.2021.00066 (cit. on pp. 3, 20, 178).

[PCH21c] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Des réfutations SAT aux réfutations Max-SAT”. In: *Journées Francophones de Programmation par Contraintes (JFPC)*. Online, France, June 2021 (cit. on pp. 4, 20, 153).

[PCH21d] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Inferring Clauses and Formulas in Max-SAT”. In: *33rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2021, Washington, DC, USA, November 1-3, 2021*. IEEE, 2021, pp. 632–639. DOI: 10.1109/ICTAI52525.2021.00101 (cit. on pp. 3, 20, 152, 177, 178).

[PCH22a] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Certificats d’optimalité pour Max-SAT”. In: *Journées Francophones de Programmation par Contraintes (JFPC)*. Saint-Étienne, France, June 2022 (cit. on pp. 4, 20, 153).

[PCH22b] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Explication de clauses et de formules dans Max-SAT”. In: *Journées Francophones de Programmation par Contraintes (JFPC)*. Saint-Étienne, France, June 2022 (cit. on pp. 4, 20, 153).

[PCH22c] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Max-réfutations et oracles SAT”. In: *Journées Francophones de Programmation par Contraintes (JFPC)*. Saint-Étienne, France, June 2022 (cit. on pp. 4, 20, 178).

[PCH22d] Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. “Proofs and Certificates for Max-SAT”. In: *J. Artif. Intell. Res.* 75 (2022), pp. 1373–1400. DOI: 10.1613/jair.1.13811 (cit. on pp. 3, 20).

[RH21] Joseph E Reeves and M Heule. “The impact of bounded variable elimination on solving pigeonhole formulas”. In: *Proceedings of Pragmatics of (SAT)* (2021) (cit. on p. 58).

[Rin21] Jussi Rintanen. “Planning and SAT”. In: *Handbook of Satisfiability - Second Edition*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, et al. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 765–789. DOI: 10.3233/FAIA201003 (cit. on p. 39).

[Rob65] John Alan Robinson. “A Machine-Oriented Logic Based on the Resolution Principle”. In: *J. ACM* 12.1 (1965), pp. 23–41. DOI: 10.1145/321250.321253 (cit. on pp. 20, 43, 58, 69–71, 88, 89, 112, 113, 151).

[Rol02] Andrea Roli. “Criticality and Parallelism in Structured SAT Instances”. In: *Principles and Practice of Constraint Programming - CP 2002*. Ed. by Pascal Van Hentenryck. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 714–719. ISBN: 978-3-540-46135-7 (cit. on p. 66).

[RBW06] Francesca Rossi, Peter van Beek, and Toby Walsh, eds. *Handbook of constraint programming*. en. Foundations of artificial intelligence. London, England: Elsevier Science, Aug. 2006 (cit. on pp. 80, 181).

[RM21] Olivier Roussel and Vasco Manquinho. “Pseudo-Boolean and cardinality constraints”. In: *Handbook of satisfiability*. IOS Press, 2021, pp. 1087–1129 (cit. on p. 100).

[RN20] Stuart Russell and Peter Norvig. *Artificial intelligence*. 4th ed. Upper Saddle River, NJ: Pearson, Nov. 2020 (cit. on p. 75).

[Rya04] Lawrence Ryan. "Efficient algorithms for clause-learning SAT solvers". MA thesis. Simon Fraser University, 2004 (cit. on pp. 53, 57, 122).

[Saf+07] Sean Safarpour, Hratch Mangassarian, Andreas G. Veneris, et al. "Improved Design Debugging Using Maximum Satisfiability". In: *Formal Methods in Computer-Aided Design, 7th International Conference, FMCAD 2007, Austin, Texas, USA, November 11-14, 2007, Proceedings*. IEEE Computer Society, 2007, pp. 13–19. DOI: 10.1109/FAMCAD.2007.26 (cit. on p. 85).

[SBJ16] Paul Saikko, Jeremias Berg, and Matti Järvisalo. "LMHS: A SAT-IP Hybrid MaxSAT Solver". In: *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. Ed. by Nadia Creignou and Daniel Le Berre. Vol. 9710. Lecture Notes in Computer Science. Springer, 2016, pp. 539–546. DOI: 10.1007/978-3-319-40970-2_34 (cit. on p. 110).

[Sch+95] John S Schlipf, Fred S Annexstein, John V Franco, et al. "On finding solutions for extended Horn formulas". In: *Information Processing Letters* 54.3 (1995), pp. 133–137. ISSN: 0020-0190. DOI: 10.1016/0020-0190(95)00019-9 (cit. on pp. 38, 67).

[Sch80] A. Schönhage. "Storage Modification Machines". In: *SIAM Journal on Computing* 9.3 (1980), pp. 490–508. DOI: 10.1137/0209036 (cit. on p. 27).

[Sch18] Christian Schulte. "Programming branchers". In: *Modeling and Programming with Gecode*. Ed. by Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. Corresponds to Gecode 6.0.1. 2018 (cit. on pp. 80, 120).

[SS01] Dale Schuurmans and Finnegan Southey. "Local search characteristics of incomplete SAT procedures". In: *Artificial Intelligence* 132.2 (2001), pp. 121–150. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(01)00151-5.

[SSH01] Dale Schuurmans, Finnegan Southey, and Robert C. Holte. "The Exponentiated Subgradient Algorithm for Heuristic Boolean Programming". In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'01. Seattle, WA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 334–341. ISBN: 1558608125 (cit. on p. 65).

[Seg07] Nathan Segerlind. "The Complexity of Propositional Proofs". In: *Bulletin of Symbolic Logic* 13.4 (2007), pp. 417–481. DOI: 10.2178/bsl/1203350879 (cit. on pp. 71, 73).

[SK93] Bart Selman and Henry Kautz. "Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems". In: *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*. IJCAI'93. Chambery, France: Morgan Kaufmann Publishers Inc., 1993, pp. 290–295 (cit. on p. 65).

[SKC94] Bart Selman, Henry A. Kautz, and Brain Cohen. "Noise Strategies for Improving Local Search". In: AAAI'94. Seattle, Washington: AAAI Press, 1994, pp. 337–343 (cit. on pp. 60, 62, 111).

[SLM92] Bart Selman, Hector Levesque, and David Mitchell. "A New Method for Solving Hard Satisfiability Problems". In: *Proceedings of the Tenth National Conference on Artificial Intelligence*. AAAI'92. San Jose, California: AAAI Press, 1992, pp. 440–446. ISBN: 0-262-51063-4 (cit. on pp. 60, 61).

[SB19] Daniel Selsam and Nikolaj Bjørner. "Guiding High-Performance SAT Solvers with Unsat-Core Predictions". In: *Theory and Applications of Satisfiability Testing – SAT 2019*. Ed. by Mikoláš Janota and Inês Lynce. Cham: Springer International Publishing, 2019, pp. 336–353. ISBN: 978-3-030-24258-9 (cit. on p. 80).

[Sem+22] Alexander Semenov, Artem Pavlenko, Daniil Chivilikhin, et al. "On Probabilistic Generalization of Backdoors in Boolean Satisfiability". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.9 (2022), pp. 10353–10361. DOI: 10.1609/aaai.v36i9.21277 (cit. on p. 67).

[SA20] Ahmad Shabani and Bijan Alizadeh. “PMTP: A MAX-SAT-Based Approach to Detect Hardware Trojan Using Propagation of Maximum Transition Probability”. In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 39.1 (2020), pp. 25–33. DOI: 10.1109/TCAD.2018.2889663 (cit. on p. 85).

[Shu12] Wang Shu-Xi. “The Improved Dijkstra’s Shortest Path Algorithm and Its Application”. In: *Procedia Engineering* 29 (2012). 2012 International Workshop on Information and Electronics Engineering, pp. 1186–1190. ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2012.01.110 (cit. on p. 25).

[Si+16] Xujie Si, Xin Zhang, Vasco M. Manquinho, et al. “On Incremental Core-Guided MaxSAT Solving”. In: *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*. Ed. by Michel Rueher. Vol. 9892. Lecture Notes in Computer Science. Springer, 2016, pp. 473–482. DOI: 10.1007/978-3-319-44953-1_30 (cit. on p. 109).

[Sie05] Wilfried Sieg. “Only Two Letters: The Correspondence between Herbrand and Gödel”. In: *The Bulletin of Symbolic Logic* 11.2 (2005), pp. 172–184. ISSN: 10798986. URL: https://www.jstor.org/stable/1556747 (cit. on p. 27).

[Sil99] João P. Marques Silva. “The Impact of Branching Heuristics in Propositional Satisfiability Algorithms”. In: *Proceedings of the 9th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence*. EPIA ’99. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 62–74. ISBN: 354066548X (cit. on p. 50).

[Sil10] João P. Marques Silva. “Minimal Unsatisfiability: Models, Algorithms and Applications (Invited Paper)”. In: *40th IEEE International Symposium on Multiple-Valued Logic, ISMVL 2010, Barcelona, Spain, 26-28 May 2010*. IEEE Computer Society, 2010, pp. 9–14. DOI: 10.1109/ISMVL.2010.11 (cit. on p. 109).

[SS96] João P. Marques Silva and Karem A. Sakallah. “GRASP - a new search algorithm for satisfiability”. In: *Proceedings of the 1996 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1996, San Jose, CA, USA, November 10-14, 1996*. Ed. by Rob A. Rutenbar and Ralph H. J. M. Otten. IEEE, 1996, pp. 220–227. DOI: 10.1109/ICCAD.1996.569607 (cit. on pp. 42, 44, 46, 47, 50, 56, 91, 96).

[Sin05] Carsten Sinz. “Towards an Optimal CNF Encoding of Boolean Cardinality Constraints”. In: *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*. Ed. by Peter van Beek. Vol. 3709. Lecture Notes in Computer Science. Springer, 2005, pp. 827–831. DOI: 10.1007/11564751_73 (cit. on p. 106).

[SHS03] Kevin Smyth, Holger H. Hoos, and Thomas Stützle. “Iterated Robust Tabu Search for MAX-SAT”. In: *Advances in Artificial Intelligence, 16th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2003, Halifax, Canada, June 11-13, 2003, Proceedings*. Ed. by Yang Xiang and Brahim Chaib-draa. Vol. 2671. Lecture Notes in Computer Science. Springer, 2003, pp. 129–144. DOI: 10.1007/3-540-44886-1_12 (cit. on p. 111).

[Son16] Tomohiro Sonobe. “Looking Inside Literal Blocks: Towards Mining More Promising Learnt Clauses in SAT Solving”. In: *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. 2016, pp. 972–979. DOI: 10.1109/ICTAI.2016.0150 (cit. on p. 56).

[SNC09a] Mate Soos, Karsten Nohl, and Claude Castelluccia. “Extending SAT Solvers to Cryptographic Problems”. In: *Theory and Applications of Satisfiability Testing - SAT 2009*. Ed. by Oliver Kullmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 244–257. ISBN: 978-3-642-02777-2 (cit. on p. 42).

[SNC09b] Mate Soos, Karsten Nohl, and Claude Castelluccia. “Extending SAT Solvers to Cryptographic Problems”. In: *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*. Ed. by Oliver Kullmann. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009, pp. 244–257. DOI: 10.1007/978-3-642-02777-2_24 (cit. on p. 39).

[SB09] Niklas Sörensson and Armin Biere. “Minimizing Learned Clauses”. In: *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*. Ed. by Oliver Kullmann. Vol. 5584. Lecture Notes in Computer Science. Springer, 2009, pp. 237–243. DOI: 10.1007/978-3-642-02777-2_23 (cit. on p. 56).

[Spe93] William M Spears. “Simulated annealing for hard satisfiability problems.” In: *Cliques, Coloring, and Satisfiability* 26 (1993), pp. 533–558 (cit. on p. 66).

[SSS97] Olaf Steinmann, Antje Strohmaier, and Thomas Stützlel. “Tabu search vs. random walk”. In: *Annual Conference on Artificial Intelligence*. Springer. 1997, pp. 337–348 (cit. on pp. 62, 63).

[SP05] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan. “NiVER: Non-increasing Variable Elimination Resolution for Preprocessing SAT Instances”. In: *Theory and Applications of Satisfiability Testing*. Ed. by Holger H. Hoos and David G. Mitchell. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 276–291. ISBN: 978-3-540-31580-3 (cit. on p. 58).

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. Cambridge, MA, USA: MIT Press, 1998. ISBN: 978-0-262-19398-6. URL: https://www.worldcat.org/oclc/37293240 (cit. on pp. 19, 54, 75, 77, 80, 132).

[Tho33] William R. Thompson. “On the likelihood that one unknown probability exceeds another in view of the evidence of two samples”. In: *Biometrika* 25.3-4 (1933), pp. 285–294. ISSN: 0006-3444. DOI: 10.1093/biomet/25.3-4.285 (cit. on p. 77).

[Tho06] John Thornton. “Clause Weighting Local Search for SAT”. In: *SAT 2005*. Ed. by Enrico Giunchiglia and Toby Walsh. Dordrecht: Springer Netherlands, 2006, pp. 97–142. ISBN: 978-1-4020-5571-3 (cit. on pp. 65, 66).

[Tho+04] John Thornton, Duc Nghia Pham, Stuart Bain, et al. “Additive versus Multiplicative Clause Weighting for SAT”. In: AAAI’04. San Jose, California: AAAI Press, 2004, pp. 191–196. ISBN: 0262511835 (cit. on p. 65).

[Tok10] Michel Tokic. “Adaptive ϵ-Greedy Exploration in Reinforcement Learning Based on Value Differences”. In: *KI 2010: Advances in Artificial Intelligence*. Ed. by Rüdiger Dillmann, Jürgen Beyerer, Uwe D. Hanebeck, et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 203–210 (cit. on p. 77).

[Tra+10] Long Tran-Thanh, Archie Chapman, Enrique Munoz de Cote, et al. “ϵ-First Policies for Budget-Limited Multi-Armed Bandits”. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. AAAI’10. Atlanta, Georgia: AAAI Press, 2010, pp. 1211–1216 (cit. on p. 77).

[Tse83] G. S. Tseitin. “On the Complexity of Derivation in Propositional Calculus”. In: *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Ed. by Jörg H. Siekmann and Graham Wrightson. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 466–483. ISBN: 978-3-642-81955-1. DOI: 10.1007/978-3-642-81955-1_28 (cit. on pp. 35, 68, 72, 73).

[Tur37a] A. M. Turing. “Computability and λ-Definability”. In: *The Journal of Symbolic Logic* 2.4 (1937), pp. 153–163. ISSN: 00224812. URL: https://www.jstor.org/stable/2268280 (cit. on p. 27).

[Tur37b] A. M. Turing. “On Computable Numbers, with an Application to the Entscheidungsproblem”. In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pp. 230–265. DOI: https://doi.org/10.1112/plms/s2-42.1.230 (cit. on pp. 27, 28).

[Tur48] Alan Turing. “Intelligent machinery: a report”. In: *London: National Physical Laboratory* (1948) (cit. on p. 27).

[Urq95] Alasdair Urquhart. “The Complexity of Propositional Proofs”. In: *Bull. Symbolic Logic* 1.4 (1995), pp. 425–467. URL: https://projecteuclid.org:443/euclid.bsl/1181154880 (cit. on p. 73).

[Urq11] Alasdair Urquhart. “A Near-Optimal Separation of Regular and General Resolution”. In: *SIAM J. Comput.* 40.1 (2011), pp. 107–121. ISSN: 0097-5397. DOI: 10.1137/090772897 (cit. on p. 73).

[Vae+20] Pashootan Vaezipoor, Gil Lederman, Yuhuai Wu, et al. “Learning Clause Deletion Heuristics with Reinforcement Learning”. In: *5th Conference on Artificial Intelligence and Theorem Proving*. 2020 (cit. on p. 80).

[Val+13] Michal Valko, Nathaniel Korda, Rémi Munos, et al. “Finite-Time Analysis of Kernelised Contextual Bandits”. In: *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013*. Ed. by Ann E. Nicholson and Padhraic Smyth. AUAI Press, 2013. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1%5C&smnu=2%5C&article%5C_id=2427%5C&proceeding%5C_id=29 (cit. on p. 79).

[Val+20] Vincent Vallade, Ludovic Le Frioux, Souheib Baarir, et al. “Community and LBD-Based Clause Sharing Policy for Parallel SAT Solving”. In: *Theory and Applications of Satisfiability Testing – SAT 2020*. Ed. by Luca Pulina and Martina Seidl. Cham: Springer International Publishing, 2020, pp. 11–27. ISBN: 978-3-030-51825-7 (cit. on p. 59).

[Vin+20] Marc Vinyals, Jan Elffers, Jan Johannsen, et al. “Simplified and Improved Separations Between Regular and General Resolution by Lifting”. In: *Theory and Applications of Satisfiability Testing – SAT 2020*. Ed. by Luca Pulina and Martina Seidl. Cham: Springer International Publishing, 2020, pp. 182–200. ISBN: 978-3-030-51825-7 (cit. on p. 73).

[Wal97] Toby Walsh. “Depth-bounded discrepancy search”. In: *IJCAI*. Vol. 97. 1997, pp. 1388–1393.

[Wal99] Toby Walsh. “Search in a Small World”. In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI’99. Stockholm, Sweden: Morgan Kaufmann Publishers Inc., 1999, pp. 1172–1177. URL: https://www.ijcai.org/Proceedings/99-2/Papers/071.pdf (cit. on pp. 57, 68).

[Wat+20] Hugues Wattez, Frederic Koriche, Christophe Lecoutre, et al. “Learning Variable Ordering Heuristics with Multi-Armed Bandits and Restarts”. In: *Proceedings of ECAI*. 2020 (cit. on pp. 80, 120, 122, 132).

[WGG11] Richard Weber, Kevin Glazebrook, and John C Gittins. *Multi-armed Bandit Allocation Indices*. en. 2nd ed. Hoboken, NJ: Wiley-Blackwell, Mar. 2011 (cit. on p. 75).

[WLZ08] Wanxia Wei, Chu Min Li, and Harry Zhang. “Switching among Non-Weighting, Clause Weighting, and Variable Weighting in Local Search for SAT”. In: *Principles and Practice of Constraint Programming*. Ed. by Peter J. Stuckey. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 313–326. ISBN: 978-3-540-85958-1 (cit. on p. 65).

[WHH14] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. “DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs”. In: *Theory and Applications of Satisfiability Testing – SAT 2014*. Ed. by Carsten Sinz and Uwe Egly. Cham: Springer International Publishing, 2014, pp. 422–429. ISBN: 978-3-319-09284-3 (cit. on p. 72).

[WQL09] Guneshi T. Wickramaarachchi, Wahbeh H. Qardaji, and Ninghui Li. “An efficient framework for user authorization queries in RBAC systems”. In: *14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, Stresa, Italy, June 3-5, 2009, Proceedings*. Ed. by Barbara Carminati and James Joshi. ACM, 2009, pp. 23–32. DOI: 10.1145/1542207.1542213 (cit. on p. 85).

[WGS03] Ryan Williams, Carla P. Gomes, and Bart Selman. “Backdoors to Typical Case Complexity”. In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI’03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003, pp. 1173–1178 (cit. on pp. 51, 67).

[WGS13] Andreas Wotzlaw, Alexander van der Grinten, and Ewald Speckenmeyer. “Effectiveness of pre- and inprocessing for CDCL-based SAT solving”. In: *CoRR* abs/1310.4756 (2013). URL: http://arxiv.org/abs/1310.4756 (cit. on p. 58).

[WW00] Zhe Wu and Benjamin W. Wah. “An Efficient Global-Search Strategy in Discrete Lagrangian Methods for Solving Hard Satisfiability Problems”. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*. AAAI Press, 2000, pp. 310–315. ISBN: 0262511126 (cit. on p. 65).

[XY18] Wei Xia and Roland H. C. Yap. “Learning Robust Search Strategies Using a Bandit-Based Approach”. In: *Proceedings of AAAI*. 2018, pp. 6657–6665 (cit. on pp. 80, 120, 122, 132).

[Xia+19a] Fan Xiao, Chu-Min Li, Mao Luo, et al. “A branching heuristic for SAT solvers based on complete implication graphs”. In: *Sci. China Inf. Sci.* 62.7 (2019), 72103:1–72103:13. DOI: 10.1007/s11432-017-9467-7 (cit. on pp. 42, 52, 55).

[Xia+19b] Fan Xiao, Chu-Min Li, Mao Luo, et al. “A branching heuristic for SAT solvers based on complete implication graphs”. In: *Science China Information Sciences* 62.7 (2019), p. 72103. ISSN: 1869-1919. DOI: 10.1007/s11432-017-9467-7 (cit. on p. 55).

[XZ05] Zhao Xing and Weixiong Zhang. “MaxSolver: An efficient exact algorithm for (weighted) maximum satisfiability”. In: *Artif. Intell.* 164.1-2 (2005), pp. 47–80. DOI: 10.1016/j.artint.2005.01.004 (cit. on p. 87).

[XRS03] Hui Xu, Rob A. Rutenbar, and Karem A. Sakallah. “sub-SAT: a formulation for relaxed Boolean satisfiability with applications in routing”. In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 22.6 (2003), pp. 814–820. DOI: 10.1109/TCAD.2003.811450 (cit. on pp. 85, 101).

[XL99] Ke Xu and Wei Li. “The SAT phase transition”. In: *Science in China Series E: Technological Sciences* 42.5 (1999), pp. 494–501 (cit. on p. 66).

[YI98] Mutsunori Yagiura and Toshihide Ibaraki. “Efficient 2 and 3-Flip Neighborhood Search Algorithms for the MAX SAT”. In: *Computing and Combinatorics, 4th Annual International Conference, COCOON ’98, Taipei, Taiwan, R.o.C., August 12-14, 1998, Proceedings*. Ed. by Wen-Lian Hsu and Ming-Yang Kao. Vol. 1449. Lecture Notes in Computer Science. Springer, 1998, pp. 105–116. DOI: 10.1007/3-540-68535-9_14 (cit. on p. 111).

[YI01] Mutsunori Yagiura and Toshihide Ibaraki. “Efficient 2 and 3-Flip Neighborhood Search Algorithms for the MAX SAT: Experimental Evaluation”. In: *J. Heuristics* 7.5 (2001), pp. 423–442. DOI: 10.1023/A:1011306011437 (cit. on p. 111).

[AMM22b] Tasniem Al-Yahya, Mohamed El Bachir Menai, and Hassan Mathkour. “Boosting the Performance of CDCL-Based SAT Solvers by Exploiting Backbones and Backdoors”. In: *Algorithms* 15.9 (2022). ISSN: 1999-4893. DOI: 10.3390/a15090302 (cit. on p. 67).

[Yan94] M. Yannakakis. “On the Approximation of Maximum Satisfiability”. In: *Journal of Algorithms* 17.3 (1994), pp. 475–502. ISSN: 0196-6774. DOI: https://doi.org/10.1006/jagm.1994.1045 (cit. on p. 112).

[Yan92] Mihalis Yannakakis. “On the Approximation of Maximum Satisfiability”. In: *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA ’92. Orlando, Florida, USA: Society for Industrial and Applied Mathematics, 1992, pp. 1–9. ISBN: 089791466X (cit. on p. 112).

[YP19] Emre Yolcu and Barnabás Póczos. “Learning Local Search Heuristics for Boolean Satisfiability”. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2019 (cit. on p. 80).

[ZM88] Ramin Zabih and David McAllester. “A Rearrangement Search Strategy for Determining Propositional Satisfiability”. In: *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*. AAAI'88. Saint Paul, Minnesota: AAAI Press, 1988, pp. 155–160 (cit. on pp. 50, 87).

[Zha97] Hantao Zhang. “SATO: An efficient propositional prover”. In: *Automated Deduction—CADE-14*. Ed. by William McCune. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 272–275. ISBN: 978-3-540-69140-2 (cit. on p. 47).

[Zha21] Hantao Zhang. “Combinatorial Designs by SAT Solvers”. In: *Handbook of Satisfiability - Second Edition*. Ed. by Armin Biere, Marijn Heule, Hans van Maaren, et al. Vol. 336. Frontiers in Artificial Intelligence and Applications. IOS Press, 2021, pp. 819–858. DOI: 10.3233/FAIA201005 (cit. on p. 39).

[ZBH96] Hantao Zhang, Maria Paola Bonacina, and Jieh Hsiang. “PSATO: A Distributed Propositional Prover and Its Application to Quasigroup Problems”. In: *J. Symb. Comput.* 21.4–6 (1996), pp. 543–560. ISSN: 0747-7171. DOI: 10.1006/jsco.1996.0030 (cit. on p. 59).

[Zha+01] Lintao Zhang, Conor F. Madigan, Matthew H. Moskewicz, et al. “Efficient Conflict Driven Learning in a Boolean Satisfiability Solver”. In: *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design*. ICCAD '01. San Jose, California: IEEE Press, 2001, pp. 279–285. ISBN: 0780372492 (cit. on pp. 42, 44).

[Zha04] Weixiong Zhang. “Configuration landscape analysis and backbone guided local search.: Part I: Satisfiability and maximum satisfiability”. In: *Artificial Intelligence* 158.1 (2004), pp. 1–26. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2004.04.001 (cit. on p. 67).

[Zha+18] Yueling Zhang, Min Zhang, Geguang Pu, et al. “Towards Backbone Computing: A Greedy-Whitening Based Approach”. In: *AI Commun.* 31.3 (2018), pp. 267–280. ISSN: 0921-7126. DOI: 10.3233/AIC-180763 (cit. on p. 67).

[ZXZ21] Zaijun Zhang, Daoyun Xu, and Jincheng Zhou. “A Structural Entropy Measurement Principle of Propositional Formulas in Conjunctive Normal Form”. In: *Entropy* 23.3 (2021). ISSN: 1099-4300. DOI: 10.3390/e23030303 (cit. on p. 69).

[Zhe+22] Jiongzhi Zheng, Kun He, Jianrong Zhou, et al. “BandMaxSAT: A Local Search MaxSAT Solver with Multi-armed Bandit”. In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*. Ed. by Lud De Raedt. International Joint Conferences on Artificial Intelligence Organization, July 2022, pp. 1901–1907. DOI: 10.24963/ijcai.2022/264 (cit. on p. 111).

[Zhu+11] Charlie Shucheng Zhu, Georg Weissenbacher, Divjyot Sethi, et al. “SAT-based techniques for determining backbones for post-silicon fault localisation”. In: *2011 IEEE International High Level Design Validation and Test Workshop, HLDVT 2011, Napa Valley, CA, USA, November 9-11, 2011*. Ed. by Zeljko Zilic and Sandeep K. Shukla. IEEE Computer Society, 2011, pp. 84–91. DOI: 10.1109/HLDVT.2011.6113981 (cit. on p. 67).

[Zhu+12] Zhu Zhu, Chu-Min Li, Felip Manyà, et al. “A New Encoding from MinSAT into MaxSAT”. In: *Principles and Practice of Constraint Programming (CP)*. Ed. by Michela Milano. Berlin, Heidelberg: Springer, 2012, pp. 455–463. ISBN: 978-3-642-33558-7 (cit. on p. 85).

[Zuy11] Anke van Zuylen. “Simpler 3/4-Approximation Algorithms for MAX SAT”. In: *Approximation and Online Algorithms - 9th International Workshop, WAOA 2011, Saarbrücken, Germany, September 8-9, 2011, Revised Selected Papers*. Ed. by Roberto Solis-Oba and Giuseppe Persiano. Vol. 7164. Lecture Notes in Computer Science. Springer, 2011, pp. 188–197. DOI: 10.1007/978-3-642-29116-6_16 (cit. on p. 112).

[ZPV10] Katharina A. Zweig, Gergely Palla, and Tamás Vicsek. “What makes a phase transition? Analysis of the random satisfiability problem”. In: *Physica A: Statistical Mechanics and its Applications* 389.8 (2010), pp. 1501–1511. ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2009.12.051 (cit. on p. 66).