

Improvements of SAT Solving Techniques and Their Application to the Coalition Structure Generation Problem

査, 澳龍

<https://hdl.handle.net/2324/1959138>

出版情報 : Kyushu University, 2018, 博士 (情報科学) , 課程博士
バージョン :
権利関係 :



**Improvements of SAT Solving Techniques
and Their Application to the Coalition
Structure Generation Problem**

Zha Aolong



KYUSHU UNIVERSITY

Improvements of SAT Solving Techniques and Their Application to the Coalition Structure Generation Problem

Graduate School of Information Science and Electrical Engineering
Department of Informatics – Fujita Laboratory

Candidate

Aolong Zha

ID number 3IE15008P

Dissertation Advisors

Assoc. Prof. Hiroshi Fujita

Assist. Prof. Miyuki Koshimura

Co-Advisor

Prof. Ryuzo Hasegawa

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Informatics

July 2018

Dissertation defended on 11 July 2018
in front of a Board of Examiners composed by:
Assoc. Prof. Fujita Hiroshi (chairman)
Prof. Yokoo Makoto
Prof. Zhao Jianjun

**Improvements of SAT Solving Techniques and Their Application to the
Coalition Structure Generation Problem**

Ph.D. dissertation. Kyushu University
© 2018 Aolong Zha. All rights reserved

This dissertation has been typeset by $\text{\LaTeX}$ and the Thesis class.

Version: July 22, 2018

Author's email: CyouRyuuRyuu@gmail.com

Abstract

In mathematics and computer science, there are a lot of problems that we know how to program a computer to solve “efficiently”, such as basic arithmetic, sorting a list, searching through a data structure. These problems can be solved in polynomial time, abbreviated as $\mathcal{P}$. It means the number of assembly steps which are taken by summing two numbers, or sorting a list, grows manageably with the magnitude of the number or the size of the list. However, there is another class of problems for which it is quick to verify whether or not a possible solution to the problem is correct, but we do not know how to efficiently find a solution yet. This class of problems are said to be solvable in nondeterministic polynomial-time, abbreviated as $\mathcal{NP}$. It is self-evident that any problem in $\mathcal{P}$ is automatically in $\mathcal{NP}$, while to date, no one can answer the question whether or not the reverse is true. If there exists such an algorithm that could solve $\mathcal{NP}$ problems in polynomial time, it would have mind-blowing implications throughout most of mathematics, science and technology.

The propositional satisfiability (SAT) problem is the problem of determining if there exists an interpretation that satisfies a given propositional formula. SAT is also the first problem that was proven to be $\mathcal{NP}$ -complete—the “hardest” problems in $\mathcal{NP}$. Various problems in artificial intelligence and computer science such as circuit/network verification, scheduling problem and constraint satisfaction problem, etc. can be reduced to SAT. Therefore, SAT problems has been emphasized for a long time and scrutinized. SAT is also interpreted in a rather broad sense. Besides plain propositional satisfiability, it includes the domains of maximum satisfiability (MaxSAT), pseudo-Boolean (PB) constraints and so on. The MaxSAT problem, an extension of SAT, is a natural combinatorial optimization problem. The essence of MaxSAT is trying to satisfy the maximum number of the constraints of a given SAT instance in the case that not all of these constraints can be satisfied. In addition, encoding PB constraints is an indispensable theory for expressing combinatorial optimization problems which can be formulated as 0–1 linear programming, and a significant effort has been devoted to the development of good SAT encoding techniques for these constraints.

The first objective of this dissertation is to speed up SAT solver. There are unit propagation and pure literal elimination (PLE) as the deduction engines of SAT solver. However, due to the data structure, detection of pure literals takes more time than detection of unit clauses, modern solvers do PLE only in preprocessing. On the other hand, it is a feature of modern solvers to make failure learning to avoid repeating the identical contradiction when discovering inconsistency during search. Existing methods learn one conflict reason per contradiction, but generally it is possible to detect multiple conflict reasons. In this dissertation, we focus on PLE strategy, conflict detection policies, learning clause management in SAT solver, and propose two new methods to solve SAT problems more efficiently.

The enhancement of MaxSAT solver’s performance by improving the conjunctive normal form (CNF) encodings of PB constraints is the second purpose of this dissertation. The existing encodings suffer from memory-out and/or execution slow down caused by an enormous size of the encoded formula. We consider a new encoding method, which applies multiple modular arithmetic in the mixed radix base,

and we also give a tailored heuristics of the base selection for modular arithmetic. In contrast, our method achieves a compact space complexity by significantly reducing the number of auxiliary variables and clauses. Theoretically, we prove the correctness of our encoding and show that it always produces a polynomial-size CNF with respect to the number of input variables.

As the third intention in this dissertation, we practically propose one of the effective applications of MaxSAT to a well-known combinatorial optimization problem, called the coalition structure generation (CSG) problem which relates to cooperative game theory and multi-agent systems. A coalition of agents can sometimes accomplish things that individual agents cannot do or perhaps do them more efficiently. The CSG problem is one major research topic in coalitional games—abstract models of cooperation, which involves partitioning a set of agents into coalitions to maximize social surplus, that is, the sum of the rewards of all coalitions. This dissertation handles CSG problems for partition function games, and uses partition decision trees (PDTs) as a compact representation for the utilities of possible divisions. We present and compare two methods to find optimal partitions of a set of agents for solving CSG problems. One is a branch-and-bound algorithm based on depth-first search, and the other is MaxSAT encoding and leaves the rest manipulation to MaxSAT solver. In our experiment, it is shown that the latter is probably superior to the former when the PDT rules are negative. Furthermore, the latter’s standard deviation of runtime is empirically less than the former’s.

The established contributions of this dissertation are summarized as follows.

1. Speeding up modern SAT solvers by two original efficient implementations
2. Enhancing MaxSAT solver by improving CNF encoding of PB constraints
3. Demonstrating the efficiency of our MaxSAT solver through solving CSG problems

Acknowledgments

First of all, I would like to express my sincere gratitude to my dissertation advisors, Miyuki Koshimura and Hiroshi Fujita for their valuable suggestions.

I am deeply indebted to Japan Student Services Organization and Kyushu University, for giving me financial support which was indispensable to my living.

Besides, I am grateful to Naoki Uemura, Toshitaka Arimura, Xiaojuan Liao, Emiko Masunaga and all other members of Fujita Laboratory, for continuously encourage me, and giving various support to my research.

I would also like to acknowledge my collaborators, Kazuki Nomoto, Yuko Sakurai, Suguru Ueda and Makoto Yokoo for providing me with an opportunity to work on a cooperative research.

My special appreciations go to professor Jianjun Zhao for his precious comments.

Last, but not least, I would like to thank my family: Jinping, Shanjun and Qingzhou for everything; Yuhua for her devoted support; Jiajia and Yangyang for filling my life with pleasure.

Contents

1	Introduction	1
1.1	Logical System	1
1.2	Propositional Logic	2
1.2.1	Boolean algebra	2
1.2.2	Circuit satisfiability problem	3
1.2.3	Conjunctive normal form	4
1.2.4	Propositional satisfiability problem	8
1.3	Intractability	9
1.3.1	Asymptotic bounds	9
1.3.2	Complexity issues	10
1.3.3	Reductions	12
1.4	Organization	14
2	SAT Algorithms	15
2.1	The DPLL Algorithm	16
2.1.1	Unit propagation	17
2.1.2	Chronological backtracking	17
2.2	Search and Inference	19
2.2.1	Implication graph	19
2.2.2	Deriving a conflict-driven clause	20
2.2.3	Asserting clauses	20
2.2.4	Backtracking Scheme	21
2.3	Organization of CDCL Solver	22
3	Efficient Implementations	25
3.1	Pure Literal Elimination	26
3.1.1	Priorities of variable assignment	26
3.1.2	Pure literal detection	27
3.1.3	Dummy decision assignment	28
3.1.4	GLUCOSEPLE in SAT competition 2016	28
3.2	Clauses Learning and Deletion	29
3.2.1	Watched literal data structure	29
3.2.2	Breadth-extended approach	31
3.2.3	Clause deletion strategies and policies	32
3.2.4	GLUHACK in SAT competition 2018	33

4	Pseudo-Boolean Constraint Encoding for MaxSAT	35
4.1	Maximum Satisfiability Problem	35
4.2	QMAXSAT solver	36
4.3	Totalizer Based Encoding	39
4.3.1	Totalizer	39
4.3.2	Weighted totalizer	43
4.4	N -level Modulo Totalizer Encoding	45
4.4.1	Single-level modulo	45
4.4.2	Multi-level modulo	49
4.4.3	Selecting suitable mixed radix base	52
4.5	Implementation and Evaluation	54
4.6	Related Works	58
5	Practical Problems	61
5.1	Related Works	62
5.2	Preliminaries	62
5.3	Partition Decision Trees	64
5.4	Branch-and-Bound Algorithm	65
5.4.1	Naive algorithm for positive rules	66
5.4.2	Algorithm using modified PDTs	66
5.5	MaxSAT Encoding	68
5.6	Experimental Results	69
6	Conclusions and Future Work	75

List of Tables

1.1	Cross-reference table between syntax and semantics	1
1.2	The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record it by an ∞ symbol.	10
3.1	The correspondence of indexes between literal and variable.	26
3.2	The initialization of the array <code>occur</code> for the instance in Formula 3.1.	28
4.1	Encoding PB constraint $147x_1 + 540x_2 + 1365x_3 < 2018$ by 4-level MTO with vectors of moduli $\Lambda_1 = \langle 5, 5, 5, 5 \rangle$ and $\Lambda_2 = \langle 3, 7, 5, 6 \rangle$	53
4.2	Average runtimes and number of instances solved	56
5.1	{MPDT, MaxSAT} for {uniform, NDCS} distribution with 100 agents	71
5.2	{MPDT, MaxSAT} for {uniform, NDCS} distribution with 200 agents	72

List of Figures

1.1	A circuit with three inputs, two additional sources that have assigned truth values, and one output.	4
1.2	A mind map drawing the range of SAT-based techniques in different development directions	9
2.1	A search tree of enumerating all truth assignments over Boolean variables x_1, x_2 and x_3 in decision variable order $\overrightarrow{x_1 x_2 x_3}$ which corresponds the decision levels from top to bottom in the leftmost of figure.	16
2.2	A termination tree. Assignments shown next to nodes are derived using UP.	19
2.3	Two cuts in an implication graph, and an example of a UIP.	20
3.1	A basic abstract data type <code>vec<T></code> used throughout the code.	25
3.2	A demonstration of the watched literal data structure for clause $(x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee \neg x_5)$. After initialization ($\mathfrak{A}$), two watched literal references (pointers w_1 and w_2) keep updating (from $\mathfrak{B}$ to $\mathfrak{D}$), until the clause is declared unit ($\mathfrak{E}$). Then a conflict occurs via UP ($\mathfrak{F}$), and no reference need to move (undo) during backtracking ($\mathfrak{G}$). . . .	30
3.3	An abridged implication graph illustrating several conflicts occurring in the same decision level.	32
3.4	A glimpse of the watch list (only corresponding clauses from C_1 to C_9) on the brink of backtracking in Example 9, also illustrated in Figure 3.3. C_1 is the sequence of assignment which is duplicated from $\mathfrak{F}$ in Figure 3.2. The conflicts 1 and 2 (resp. conflicts 3 and 4) belong to the conflict pattern 1 (resp. pattern 2).	33
4.1	TO encoding: (a) Overview over binary tree of TO adders showing connection of input and output between each recursion. (b) Detailed adder for two input vectors of Boolean variables.	42
4.2	Comparison of cumulative <code>#Instances</code> solved in generated <code>#variables</code> and <code>#clauses</code>	57
4.3	Comparison of cumulative <code>#Instances</code> solved within runtime	58
4.4	Comparison of runtimes of each instance solved with n -level MTO and MaxHS	58
5.1	PDTs of coalitional game given in Example 19	65

5.2	Modified PDTs of coalitional game given in Example 19	67
5.3	A search tree for the modified PDTs in Figure 5.2	67
5.4	Uniform vs. NDCS in MPDT	73
5.5	Uniform vs. NDCS in MaxSAT	73

List of Algorithms

1	DPLL(CNF $\mathcal{F}$): returns a set of literals or unsat	18
2	CDCL(CNF $\mathcal{F}$): returns sat or unsat	22
3	The occurrence of each literal in corresponding clause(s).	27
4	Pure literal detection algorithm in solving process	29
5	QMAXSAT($\mathcal{F}_w^b$): return k_{opt}	38
6	TO($\langle(w_1, b_1) \dots (w_m, b_m)\rangle$): return $(\mathcal{Z}_{sum}, \Phi)$	41
7	WTO($\langle(w_1, b_1) \dots (w_m, b_m)\rangle$): return $(\mathcal{Z}_{sum}, \Phi)$	44
8	1LVMTO($\langle(w_1, b_1) \dots (w_m, b_m)\rangle, p$): return $(\mathcal{U}_z _p \mathcal{L}_z, \Phi)$	48
9	n LVMTO($\langle(w_1, b_1) \dots (w_m, b_m)\rangle, \Lambda$): return $(\mathcal{D}_z^{n+1} _{\lambda_n} \dots _{\lambda_1} \mathcal{D}_z^1, \Phi)$	51
10	BASESELECTION($\langle(w_1, b_1) \dots (w_m, b_m)\rangle, \kappa$): return Λ	55

Chapter 1

Introduction

1.1 Logical System

Among the important properties that logical systems can have:

- *Validity*, which means that an argument never allows a false conclusion (specifically, inference) from true premises.
- *Consistency*, which means that no theorem of the system contradicts another.
- *Completeness*, which means that if an argument is true, it can be proven to be a theorem of the system.

These properties are interdefinable if we make use of negation (in symbol $\neg$): the argument from $\phi_1, \dots, \phi_n$ to ψ is valid if and only if the set $\{\phi_1, \dots, \phi_n, \neg\psi\}$ is inconsistent, which can be understood as if it would be impossible for them all to be true. Validity and consistency are really two ways of looking at the same thing and each may be described in terms of syntax or semantics in Table 1.1.

Syntax, giving rise to *proof theory*, is restricted to definitions that refer to the syntactic form (grammatical structure) of the sentences in question. Proofs are derived with respect to an *axiom system* which is defined syntactically as consisting of a set of axioms and inference rules that sanction proof steps with specified grammatical forms. In contrast, semantics, giving rise to *model theory*, studies the way sentences relate to “the world”. A sentence is said to be true if it “corresponds to the world”. The concept of *algebraic structure* is used to make precise the meaning of “corresponds to the world”. We will speak of *formulae* instead of sentences to allow

Table 1.1. Cross-reference table between syntax and semantics

<i>Logic</i>	Syntax	Semantics
<i>Validity</i>	Derivability	Validity
<i>Consistency</i>	Consistency	Satisfiability
<i>Theory</i>	Proof theory	Model theory
<i>System</i>	Axiom system	Formula system
<i>Structure</i>	Grammatical structure	Algebraic structure
<i>Consequence</i>	Syntactic entailment $\vdash$	Semantic entailment $\models$

for the possibility that a sentence includes free variables. The customary notation is to use $\mathcal{A} \models \phi$ to say ϕ is true in the structure $\mathcal{A}$ (premise). Intuitively, an argument is valid if whenever the premises are true, so is the conclusion. Satisfiability is the semantic version of consistency. A set of *formulae* is said to be *satisfiable* if there is some structure in which all its component formulae are true: that is, $\{\phi_1, \dots, \phi_n\}$ is satisfiable if and only if, for some $\mathcal{A}$, $\mathcal{A} \models \phi_1$ and $\dots$ and $\mathcal{A} \models \phi_n$. It follows from the definitions that validity and satisfiability are mutually definable: $\phi_1, \dots, \phi_n \models \psi$ if and only if $\{\phi_1, \dots, \phi_n, \neg\psi\}$ is *unsatisfiable*.

There is a further pair of basic logical ideas closely related to satisfiability: *necessity* and *possibility*. Semantically, a sentence is said to be necessary (necessarily true) if it satisfies all models, and possible (possibly true) if it satisfies at least one model. If we understand a model (*possible world*) to be a structure, possibility turns out to be just another name for satisfiability. In logic, the technical name for a necessary formula is *tautology* ($\top$): ϕ is defined to be a *logical truth* (in symbols, $\models \phi$) if and only if, *for all* $\mathcal{A}$, $\mathcal{A} \models \phi$. There is also a syntactic version of a logical truth, namely a theorem-in-an-axiom-system. We say ϕ is a theorem of the system if and only if ϕ is derivable from the axioms alone. In a complete system, theorem-hood and logical truth coincide: $\vdash \phi$ iff $\models \phi$. Therefore, in logical truth and theorem-hood we encounter yet another pair of syntactic and semantic concepts that, although they have quite different sorts of definitions, nevertheless coincide exactly. Moreover, a formula is necessary if it is not possibly not true. In other words, $\models \phi$ if and only if it is not the case that ϕ is unsatisfiable. Ergo, satisfiability, theorem-hood, logical truths and necessities are mutually “characterizable”.

This review shows how closely related satisfiability is to the central concepts of logic. Indeed, relative to a complete axiom system, satisfiability may be used to define, and may be defined by, the other basic concepts of the field—validity, consistency, completeness, logical truth, tautology, and theorem-hood.

1.2 Propositional Logic

Propositional logic is a logical system that is intimately connected to Boolean algebra. Many syntactic concepts of Boolean algebra carry over to propositional logic with only minor changes in notation and terminology, while the semantics of propositional logic are defined via Boolean algebras in a way that the tautologies (theorems) of propositional logic correspond to equational theorems of Boolean algebra.

1.2.1 Boolean algebra

George Boole (1815-1864) advanced the state of logic considerably with the introduction of the algebra structure that bears his name: a structure $\mathcal{A} = \langle B, \vee, \wedge, \neg, 0, 1 \rangle$ is a Boolean algebra if and only if $\vee$ and $\wedge$ are binary operations and $\neg$ is a unary operation on domain B under which B is closed, $1, 0 \in B$, and satisfies the following rules and laws:

$$\text{Commutation} \begin{cases} x \vee y = y \vee x \\ x \wedge y = y \wedge x \end{cases} \quad \text{Identity} \begin{cases} x \vee 0 = x \\ x \wedge 1 = x \end{cases}$$

$$\begin{array}{ll}
\textbf{Absorption} \begin{cases} x \vee (x \wedge y) = x \\ x \wedge (x \vee y) = x \end{cases} & \textbf{Annihilation} \begin{cases} x \vee 1 = 1 \\ x \wedge 0 = 0 \end{cases} \\
\textbf{Association} \begin{cases} x \vee (y \vee z) = (x \vee y) \vee z \\ x \wedge (y \wedge z) = (x \wedge y) \wedge z \end{cases} & \textbf{Idempotence} \begin{cases} x \vee x = x \\ x \wedge x = x \end{cases} \\
\textbf{Distribution} \begin{cases} x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \\ x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \end{cases} & \textbf{Complementation} \begin{cases} x \vee \neg x = 1 \\ x \wedge \neg x = 0 \end{cases} \\
\textbf{De Morgan} \begin{cases} \neg x \vee \neg y = \neg(x \wedge y) \\ \neg x \wedge \neg y = \neg(x \vee y) \end{cases} & \textbf{Double negation} \neg(\neg x) = x
\end{array}$$

Boolean algebra was introduced in Boole's first book *The Mathematical Analysis of Logic*, and set forth more fully in his *An Investigation of the Laws of Thought*. His main innovation was to develop an algebra-like notation for the elementary properties of sets. He was one of the first to employ a symbolic language. Boole was not interested in axiom systems, but in the theory of inference. In his system it could be shown that the argument from $\phi_1, \dots, \phi_n$ to ψ is valid (i.e. $\phi_1, \dots, \phi_n \models \psi$) by deriving the equation ψ from equations $\phi_1, \dots, \phi_n$ by applying rules of inference, which were essentially cases of algebraic substitution.

Syntactically, every Boolean term corresponds to a propositional formula of propositional logic. The semantics of propositional logic rely on *truth assignments*. The essential idea of a truth assignment is that the propositional variables are mapped to elements of a fixed Boolean algebra, and then the truth value of a propositional formula using these letters is the element of the Boolean algebra that is obtained by computing the value of the Boolean term corresponding to the formula. A tautology is a propositional formula that is assigned truth value 1 by every truth assignment of its propositional variables to an arbitrary Boolean algebra.

These semantics permit a translation between tautologies of propositional logic and equational theorems of Boolean algebra. Every tautology ϕ of propositional logic can be expressed as the Boolean equation $\phi = 1$, which will be a theorem of Boolean algebra. Conversely every theorem $\phi = \psi$ of Boolean algebra corresponds to the tautologies $(\phi \wedge \psi) \vee (\neg\phi \wedge \neg\psi)$ and $(\neg\phi \vee \psi) \wedge (\neg\psi \vee \phi)$. If the operator $\rightarrow$, named “implies”, is in the language these last tautologies can also be written as $(\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi)$, or as two separate theorems $\phi \rightarrow \psi$ and $\psi \rightarrow \phi$.

1.2.2 Circuit satisfiability problem

There is a great idea about modern computing that is to let a combinatorial logic circuit, e.g., a *central processing unit* (CPU), calculates complex problems of the real world. To specify circuit satisfiability (CircuitSAT) problem, we need to make precise what we mean by a *circuit*. Consider the basic operators of Boolean algebra: $\vee, \wedge$, and $\neg$. Our definition of a circuit, referring to [44], is designed to represent a digital electronic circuit built out of gates that implement these operators. Thus we define a circuit K to be a labeled, directed acyclic graph such as the one shown in the example of Figure 1.1.

- The *sources* in K (the nodes with no incoming edges) are labeled either with

one of the constants 0 or 1, or with the name of a distinct Boolean variable. The nodes of the latter type will be referred to as the *inputs* to the circuit.

- Every other node is labeled with one of the Boolean operators $\vee$, $\wedge$, or $\neg$; nodes labeled with $\vee$ or $\wedge$ will have two incoming edges, and nodes labeled with $\neg$ will have one incoming edge.
- There is a single node with no outgoing edges, and it will represent the *output*: the result that is computed by the circuit.

A circuit computes a function of its inputs in the following natural way. We imagine the edges as “wires” that carry the 0/1 signal value at the nodes they emanate from. Each node other than the sources will take the values on its incoming edge(s) and apply the Boolean operator that labels it. The result of this $\vee$, $\wedge$, or $\neg$ operation will be passed along the edge(s) leaving this node. The overall value computed by the circuit will be the value computed at the output node.

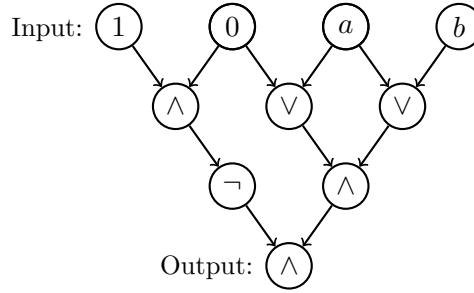


Figure 1.1. A circuit with three inputs, two additional sources that have assigned truth values, and one output.

For example, consider the circuit K in Figure 1.1. The leftmost two sources are preassigned the values 1 and 0, and the next three sources constitute the inputs. If the input Boolean variable a is assigned to value 1, then we get values 0, 1, 1 for the gates in the second row, values 1, 1 for the gates in the third row, and the value 1 for the output. Symbolically, this circuit K can be represented as a propositional logic formula:

$$\mathcal{F}_K = \neg(1 \wedge 0) \wedge ((0 \vee a) \wedge (a \vee b)) = a$$

Now, the CircuitSAT is the following decision problem. We are given a circuit K (or its propositional logic formula $\mathcal{F}_K$) as an input, and we need to decide whether there is an assignment of values to the inputs that causes the output to take the value 1. (If so, we will say that the given circuit is satisfiable, and a satisfying assignment is one that results in an output of 1.) In our example, we have just seen—via the assignment $a = 1$ —that the circuit K in Figure 1.1 is satisfiable.

1.2.3 Conjunctive normal form

Propositional logic formulae can be transformed (*encoded*) to conjunctive normal form (CNF): a conjunction of *clauses* $\bigwedge_i C_i$, each clause C_i being a disjunction of *literals* $\bigvee_j l_j$, and each literal l_j being either a Boolean variable x or its negation $\neg x$. In this dissertation, a CNF formula (resp. a clause) can also be regarded as a set of

conjunctive clauses (resp. a set of disjunctive literals). CNF has the advantage of being a very simple form, leading to easy algorithm implementation and a common file format. Unfortunately, there are several ways of encoding most combinatorial problems, and few guidelines on how to choose among them, yet the choice of encoding can be as important as the choice of search algorithm. Different encodings may have different advantages and disadvantages such as space complexity or solution density, and what is an advantage for one SAT solver might be a disadvantage for another. In short, CNF encoding is an art and we must often proceed by intuition and experimentation.

Transformation by Boolean algebra

A propositional logic formula can be transformed to a logically equivalent CNF formula by using the rules and laws of Boolean algebra.

Example 1. Consider the propositional formula

$$(a \rightarrow (c \wedge d)) \vee (b \rightarrow (c \wedge e)).$$

The implications can be decomposed:

$$((a \rightarrow c) \wedge (a \rightarrow d)) \vee ((b \rightarrow c) \wedge (b \rightarrow e)).$$

The conjunctions and disjunctions can be rearranged:

$$\begin{aligned} & ((a \rightarrow c) \vee (b \rightarrow c)) \wedge ((a \rightarrow c) \vee (b \rightarrow e)) \wedge \\ & ((a \rightarrow d) \vee (b \rightarrow c)) \wedge ((a \rightarrow d) \vee (b \rightarrow e)) \end{aligned}$$

The implications can be rewritten as disjunctions, and duplicated literals removed:

$$(\neg a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee c \vee e) \wedge (\neg a \vee \neg b \vee c \vee d) \wedge (\neg a \vee \neg b \vee d \vee e)$$

Finally, subsumed clauses can be removed, leaving the conjunction

$$(\neg a \vee \neg b \vee c) \wedge (\neg a \vee \neg b \vee d \vee e)$$

This example reduces to a compact CNF formula, but in general this method yields exponentially large formulae.

Transformation by Tseitin encoding

Conversion to CNF is more usually achieved using the well-known Tseitin encodings [70], which generate a linear number of clauses at the cost of introducing a linear number of new Boolean variables, and generate an *equisatisfiable* formula (satisfiable if and only if the original formula is satisfiable). Tseitin encodings work by adding fresh auxiliary variables to the CNF formula, one for every non-clausal subformula of the original formula, along with clauses to capture the relationships between these auxiliary variables and the subformulae.

Example 2. On the Example 1 the Tseitin encoding would introduce an auxiliary variable x with definition

$$x \leftrightarrow (c \vee d)$$

to represent subformula $(c \vee d)$. This definition can be reduced to clausal form:

$$(\neg x \vee c) \wedge (\neg x \vee d) \wedge (\neg c \vee \neg d \vee x)$$

Similarly it would introduce another definition

$$y \leftrightarrow (c \vee e)$$

which reduces to clauses

$$(\neg y \vee c) \wedge (\neg y \vee e) \wedge (\neg c \vee \neg e \vee y)$$

Applying these definitions the original formula is reduced to

$$(\neg a \vee x) \vee (\neg b \vee y)$$

which can be finally rewritten as a clausal form

$$(\neg a \vee x \vee \neg b \vee y)$$

Transformation from CNF to 3-CNF

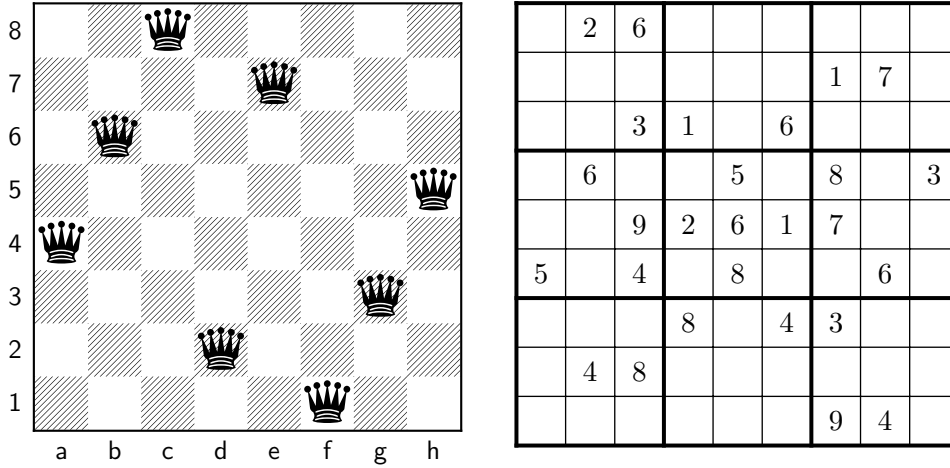
3-CNF formulae, also referred to as instances of 3-SAT problem have exactly three literals in each clause. These problems are interesting in their own right, and random 3-CNF problems have been extensively studied. A CNF formula can be transformed to a 3-CNF formula as follows. For a clause $c_i = (l_1 \vee \dots \vee l_k)$ there are four cases:

- $k = 1$** Create two new variables $b_{i,1}, b_{i,2}$ and replace c_i by the clauses $(l_1 \vee b_{i,1} \vee b_{i,2})$, $(l_1 \vee \neg b_{i,1} \vee b_{i,2})$, $(l_1 \vee b_{i,1} \vee \neg b_{i,2})$ and $(l_1 \vee \neg b_{i,1} \vee \neg b_{i,2})$.
- $k = 2$** Create one new variable $b_{i,1}$ and replace c_i by the clauses $(l_1 \vee l_2 \vee b_{i,1})$ and $(l_1 \vee l_2 \vee \neg b_{i,1})$.
- $k = 3$** Leave clause c_i unchanged.
- $k > 3$** Create new variables $b_{i,1}, b_{i,2}, \dots, b_{i,k-3}$ and replace c_i by the clauses $(l_1 \vee l_2 \vee b_{i,1})$, $(\neg b_{i,1} \vee l_3 \vee b_{i,2})$, $(\neg b_{i,2} \vee l_4 \vee b_{i,3})$, $\dots$, $(\neg b_{i,k-3} \vee l_{k-1} \vee l_k)$.

This transformation does not seem to have many practical applications, but it may be useful for generating 3-CNF benchmarks, or as a preprocessor for a specialized algorithm that only accepts instances of 3-SAT problem.

Extensional constraints in CNF

One way of modelling a problem, e.g., n -QUEENS, SUDOKU, etc., as SAT is first to model it as a constraint satisfaction problem (CSP), then to apply a standard encoding from CSP to SAT.



A finite-domain CSP has variables $v_1 \dots v_n$ each with a finite domain, and constraints prescribing prohibited combinations of values. The problem is to find an assignment of values to all variables such that no constraint is violated. The most natural and widely-used encoding from CSP to SAT is *direct encoding*. A SAT variable (i.e., Boolean variable) $x_{v,i}$, is defined as true if and only if the CSP variable v has the domain value i assigned to it. The direct encoding consists of three sets of clauses. Each CSP variable must take at least one domain value, expressed by *at-least-one* clauses:

$$\bigvee_i x_{v,i}$$

No CSP variable can take more than one domain value, expressed by *at-most-one* clauses:

$$\neg x_{v,i} \vee \neg x_{v,j}$$

Conflicts are enumerated by *conflict* clauses:

$$\neg x_{v,i} \vee \neg x_{w,j}$$

Example 3 (Graph coloring problem). Given two adjacent vertices v and w and three available colors $\{0, 1, 2\}$, each vertex must take exactly one color, but because they are adjacent they cannot take the same color. The direct encoding of the problem contains at-least-one clauses

$$(x_{v,0} \vee x_{v,1} \vee x_{v,2}) \qquad (x_{w,0} \vee x_{w,1} \vee x_{w,2})$$

at-most-one clauses

$$\begin{array}{lll} (\neg x_{v,0} \vee \neg x_{v,1}) & (\neg x_{v,0} \vee \neg x_{v,2}) & (\neg x_{v,1} \vee \neg x_{v,2}) \\ (\neg x_{w,0} \vee \neg x_{w,1}) & (\neg x_{w,0} \vee \neg x_{w,2}) & (\neg x_{w,1} \vee \neg x_{w,2}) \end{array}$$

and conflict clauses

$$(\neg x_{v,0} \vee \neg x_{w,0}) \qquad (\neg x_{v,1} \vee \neg x_{w,1}) \qquad (\neg x_{v,2} \vee \neg x_{w,2})$$

This problem has six solutions:

$$\begin{array}{ccc} [v = 0, w = 1] & [v = 0, w = 2] & [v = 1, w = 0] \\ [v = 1, w = 2] & [v = 2, w = 0] & [v = 2, w = 1] \end{array}$$

1.2.4 Propositional satisfiability problem

Definition 1 (Propositional satisfiability problem). Given a CNF formula $\mathcal{F}$ over a finite set of Boolean variables $\mathcal{X} = \{x_1, \dots, x_n\}$, the propositional satisfiability (SAT) problem is the decision problem of determining if there exists a truth assignment that satisfies $\mathcal{F} = 1$. A (partial) truth assignment is function $\mathcal{A} : X \mapsto \{0, 1\}$, where $X \subset \mathcal{X}$. This function can be extended to variables in $\mathcal{X} \setminus X$, literals, clauses and CNF formulae, resulting in a function from formulae to formulae as follows:

- For variables, $x_i \notin X$, $\mathcal{A}(x_i) = x_i$.
- For literals, $\mathcal{A}(\neg x_i) = 1 - \mathcal{A}(x_i)$, if $x_i \in X$; otherwise, $\mathcal{A}(l) = l$.
- For clauses, $\mathcal{A}(l_1 \vee \dots \vee l_r) = \mathcal{A}(l_1) \vee \dots \vee \mathcal{A}(l_r)$, and $\mathcal{A}(\square) = 0$, simplified considering $1 \vee C = 1$ and $0 \vee C = C$.
- For CNF formulae, $\mathcal{A}(C_1 \wedge \dots \wedge C_m) = \mathcal{A}(C_1) \wedge \dots \wedge \mathcal{A}(C_m)$ and $\mathcal{A}(\emptyset) = 1$, simplified considering $1 \wedge \mathcal{F} = \mathcal{F}$ and $0 \wedge \mathcal{F} = 0$.

A total truth assignment is truth assignment $\mathcal{A}$ with domain $\mathcal{X}$. Note that, dicritically, we denote the empty clause and the empty CNF formula as $\square$ and $\emptyset$ respectively. For any CNF formula, $\mathcal{A}(\mathcal{F})$ may be 0, 1, or a partial instantiation of $\mathcal{F}$, whereas for total truth assignments, $\mathcal{A}(\mathcal{F})$ is either 0 or 1. Truth assignment $\mathcal{A}$ satisfies a literal, clause, or a CNF formula if it assigns 1 to it, and *falsifies* it if it assigns 0. Thus a CNF formula is *satisfiable* if there exists a truth assignment that satisfies it. Otherwise, it is *unsatisfiable*.

There is a special case of SAT that will turn out to be equivalently difficult and is somewhat easier to think about; this is the case in which all clauses contain exactly three literals. We call this problem 3-SAT. On the other hand, SAT is special case of CircuitSAT as a CNF formula can be represented as a circuit.

SAT and 3-SAT are really fundamental combinatorial search problems; they contain the basic ingredients of a hard computational problem in very “bare-bones” fashion. We have to make n independent decisions (the assignments for each x_i) so as to satisfy a set of constraints. There are several ways to satisfy each constraint in isolation, but we have to arrange our decisions so that all constraints are satisfied simultaneously.

SAT is interpreted in a rather broad sense. Besides plain propositional satisfiability, it includes the domains of maximum satisfiability (MaxSAT) and pseudo-Boolean (PB) constraints, quantified Boolean formulae (QBF), satisfiability modulo theories (SMT), constraints programming (CSP) techniques, etc., for word-level problems

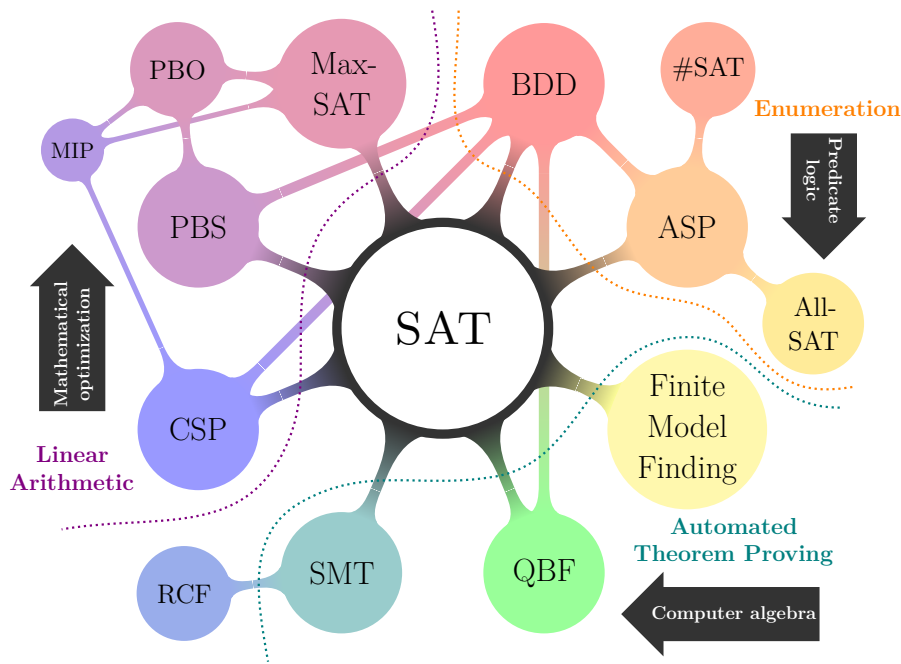


Figure 1.2. A mind map drawing the range of SAT-based techniques in different development directions

and their propositional encodings. Figure 1.2 is a mind map of paradigmatic SAT-based techniques which depicts such extended and related fields of SAT in different development directions (corresponding various problem classes).

1.3 Intractability

When people first began analyzing discrete algorithms mathematically, a consensus began to emerge on how to quantify the notion of a “reasonable” running time. The asymptotic viewpoint is inherent to computational complexity theory. In virtue of polynomials typify “slowly growing” functions, we’d like an *efficient* algorithm to have a better scaling property: when the input size increases by a factor, its running time should be only bounded by a power of this factor.

1.3.1 Asymptotic bounds

Our discussion of computational tractability has turned out to be intrinsically based on our ability to express the notion that an algorithm’s worst-case running time on inputs of size n grows at a rate that is at most proportional to some function $f(n)$. Then $f(n)$ becomes a bound on the running time of the algorithm.

Definition 2 (Asymptotic notations). Let $f(n)$ and $g(n)$ be two functions. If there exist constants (independent of n) ϵ, c such that $0 < \epsilon < c$, and $n_0 \geq 0$ so that for all $n \geq n_0$ (i.e., for sufficiently large n), we have different asymptotic notations for corresponding conditions as follows:

Table 1.2. The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record it by an ∞ symbol.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1s	< 1s	< 1s	< 1s	< 1s	< 1s	4s
$n = 30$	< 1s	< 1s	< 1s	< 1s	< 1s	18m	10^{25} y
$n = 50$	< 1s	< 1s	< 1s	< 1s	11m	36y	∞
$n = 100$	< 1s	< 1s	< 1s	1s	12,892y	10^{17} y	∞
$n = 1,000$	< 1s	< 1s	1s	18m	∞	∞	∞
$n = 10,000$	< 1s	< 1s	2m	12d	∞	∞	∞
$n = 100,000$	< 1s	2s	3h	32y	∞	∞	∞
$n = 1,000,000$	1s	20s	12d	31,710y	∞	∞	∞

O If $f(n) \leq c \cdot g(n)$, then we say that f is asymptotic upper-bounded by g , and denoted it as $f(n) = O(g(n))$;

Ω If $\epsilon \cdot g(n) \leq f(n)$, then we say that f is asymptotic lower-bounded by g , denoted as $f(n) = \Omega(g(n))$;

Θ If $\epsilon \cdot g(n) \leq f(n) \leq c \cdot g(n)$, then we say that f is asymptotic tight-bounded by g , denoted as $f(n) = \Theta(g(n))$;

Definition 3 (Primitive computational step). In Turing machine, each primitive computational step corresponds to a single instruction of machine language on a standard processor. In particular, one primitive computational step is in $O(1)$.

Definition 4 (Polynomial-time algorithm). There are constants $c > 0$ and $d > 0$ so that on every input instances of size n for an algorithm, if its running time is bounded by cn^d primitive computational steps, i.e., the algorithm is $O(n^d)$, then it is a polynomial-time algorithm.

Problems for which polynomial-time algorithms exist almost invariably turn out to have algorithms with running times proportional to very moderately growing polynomials like $n, n \log_2 n, n^2$, or n^3 . Conversely, problems for which no polynomial-time algorithm is known tend to be very difficult in practice. For example, search spaces for combinatorial problems tend to grow exponentially in the size n of the instance¹; if the instance size increases by one, the number of possibilities increases multiplicatively. Suppose that we have a processor that executes a million high-level instructions per second, and we have algorithms with running-time bounds of $n, n \log_2 n, n^2, n^3, 1.5^n, 2^n$, and $n!$. In Table 1.2, we show the running times of these algorithms (in seconds, minutes, days, or years) for inputs of size $n = 10, 30, 50, 100, 1,000, 10,000, 100,000$ and $1,000,000$.

1.3.2 Complexity issues

Much of complexity theory deals with *decision problems*. A decision problem is any arbitrary **yes/no** question on an infinite set of instances. Decision problems fall into

¹ An instance of a problem is considered as an input of an algorithm that can solve it.

sets of comparable complexity, called complexity classes. $\mathcal{P}$ and $\mathcal{NP}$ are the most well-known complexity classes, named as an acronym for “polynomial-time” and “nondeterministic polynomial-time”, respectively.

Definition 5 (Complexity class $\mathcal{P}$). $\mathcal{P}$ is the set of decision problems each of whose containing instances can be solved (**yes/no**) by a polynomial-time algorithm. Distinctively, those of **yes**-instance consist in $\mathcal{P}_{\text{yes}}$, those of **no**-instance consist in $\mathcal{P}_{\text{no}}$, and we always have $\mathcal{P} = \mathcal{P}_{\text{yes}} \cup \mathcal{P}_{\text{no}}$.

Example 4 (IS-PRIME problem). Given an integer n ($n \geq 2$), IS-PRIME problem asks whether n is a prime number or not. There exists algorithm $\text{AKS}(n)$ [4] that returns answer **yes/no** in $O(\log_2^6 n)$. Notice that no matter instance (input) n is either a prime number (in $\mathcal{P}_{\text{yes}}$) or a composite number (in $\mathcal{P}_{\text{no}}$), IS-PRIME problem belongs to $\mathcal{P}$.

Definition 6 (Complexity class $\mathcal{NP}$). $\mathcal{NP}$ is the set of decision problems each of whose including instances must have at least one **yes**-solution, and there exists a **yes**-solution can be verified by a polynomial-time algorithm (*verifier*).

Corollary 1. $\mathcal{P}_{\text{yes}} \subseteq \mathcal{NP}$

Proof. Let X be any problem in $\mathcal{P}_{\text{yes}}$. By Definition 5, there exists a polynomial-time algorithm A that solves (returns **yes**) all instances of X . We have a polynomial-time verifier $B(n, s)$ simply returns the value $A(n)$, where n is any instance of X and s is one of the **yes**-solution of n . $\square$

While the Definition 5 is *symmetric*², the the Definition 6 is fundamentally *asymmetric*. Having efficient certificates that a given object has property C , does not automatically entail having nice certificates that a given object doesn't have it. An input n is a **yes**-instance if and only if there exists a solution s so that $B(n, s) = \text{yes}$. Negating this statement, we see that an input n is a **no**-instance if and only if for all s , it is the non-solution case that $B(n, s) = \text{no}$.

Definition 7 (Complexity class $\text{co-}\mathcal{NP}$). $\text{co-}\mathcal{NP}$ is the set of decision problems whose complements are in $\mathcal{NP}$.

Example 5 (IS-COMPOSITE problem). Given an integer n ($n \geq 2$), IS-COMPOSITE problem asks whether n is a composite number or not. If there exists a **yes**-solution s which can be interpreted as n , then it can be verify by algorithm $\text{PROD}(n, s)$ in $O(1)$. Notice that IS-COMPOSITE problem belongs to $\mathcal{NP}$ only for whose instances (inputs) exactly are composite numbers and there exists such their **yes**-solutions. Alternatively, for those inputs of prime number, IS-COMPOSITE problem is in $\text{co-}\mathcal{NP}$ by Corollary 1 and Definition 7, because IS-PRIME problem (this case is in $\mathcal{P}_{\text{yes}}$) is the complement of IS-COMPOSITE problem.

Corollary 2. $\mathcal{P}_{\text{no}} \subseteq \text{co-}\mathcal{NP}$

² Having an efficient algorithm to determine if an object has a property C is equivalent to having an efficient algorithm for the complementary set $\overline{C}$.

There is a famous anecdote of a lecture by F. N. Cole, entitled “On the Factorization of Large Numbers”, at the 1903 American Mathematical Society (AMS) meeting. Without uttering a word, he went to the chalkboard, wrote

$$2^{67} - 1 = 147573952589676412927 = 193707721 \times 761838257287$$

and proceeded to perform the long multiplication of the integers on the right hand side to derive the integer on the left: Mersene’s 67-th number (which was conjectured to be prime). No one in the audience had any questions. Cole demonstrated that the number $2^{67} - 1$ is composite, and the proof is simply a nontrivial factor of it. The fact that it was extremely hard for Cole to find these factors (he said it took him “three years of Sundays”) did not affect in any way that demonstration. It is taken for granted that the written proof is *succinct* and *efficiently verifiable*, regardless how long it took to discover, which is the features we want to extract from this motivating episode.

1.3.3 Reductions

Theoretical computer scientists explore the space of computationally hard problems, eventually arriving at a mathematical characterization of a large class of them. The basic technique in this exploration is to compare the relative difficulty of different problems; we want to formally express statements like, “Problem Y is at least as hard as problem X .”

Definition 8 (Polynomial-time Turing reduction). Problem X is polynomial-time Turing reducible to problem Y if (1) arbitrary instances of X can be solved using a polynomial number of calls to a subroutine that solves arbitrary instances of Y , and (2) polynomial time outside of those subroutine calls. We call this reduction a polynomial-time Turing reduction, a.k.a, *Cook reduction*, and denote it as $X \leq_p Y$.

$X \leq_p Y$ can be understood as “ Y is at least as hard as X ”, since suppose $X \leq_p Y$, (1) if Y can be solved in polynomial time, then X can be solved in polynomial time; (2) if X cannot be solved in polynomial time, then Y cannot be solved in polynomial time. Moreover, if $X \leq_p Y$ and $Y \leq_p X$, we can say “ Y is the same hard as X ”, denoted by $X \equiv_p Y$. Notice that here ‘hard(er)’ does not mean more time-consuming, but more complex reducibility.

Definition 9 (Hardness and completeness). A problem X is called $\mathcal{Y}$ -hard if $\forall Y \in \mathcal{Y}$ we have $X \leq_p Y$. If we further have that $X \in \mathcal{Y}$ then X is called $\mathcal{Y}$ -complete.

In other words, if X is $\mathcal{Y}$ -complete, it is a hardest problem in the class $\mathcal{Y}$: if we manage to solve X efficiently, we have done so for all other problems in $\mathcal{Y}$. It is not apriori clear that a given class has any complete problems. On the other hand, a given class may have many complete problems, and by definition, they all have essentially the same complexity. If we manage to prove that *any* of them cannot be efficiently solved, then we automatically have done so for *all* of them.

Definition 10 (Complexity class $\mathcal{NP}$ -complete). $\mathcal{NP}$ -complete is the set of decision problems via the following two properties: (1) $\mathcal{NP}$ -complete $\subseteq \mathcal{NP}$; (2) $\forall X, Y$ s.t. $X \in \mathcal{NP}$, $Y \in \mathcal{NP}$ -complete, $X \leq_p Y$.

Theorem 1 (Cook (1971) [19]). *CircuitSAT belongs to $\mathcal{NP}$ -complete.*

CircuitSAT³ is a prototypical $\mathcal{NP}$ -complete problem. There is a very practical motivation in searching further $\mathcal{NP}$ -complete problems: since it is widely believed that $\mathcal{P}_{\text{yes}} \neq \mathcal{NP}$, the discovery that a problem is $\mathcal{NP}$ -complete can be taken as a strong indication that it cannot be solved in polynomial time. Once we establish first “natural” $\mathcal{NP}$ -complete problem, others fall like dominoes (transitivity of $\leq_p$ takes care of the rest). Given a new problem Y , here is the basic strategy for proving it is $\mathcal{NP}$ -complete.

1. Show that $Y \in \mathcal{NP}$.
2. Choose an $\mathcal{NP}$ -complete problem X .
3. Prove that $X \leq_p Y$.

Theorem 2. *If $X \in \mathcal{NP}$ -complete, $Y \in \mathcal{NP}$ with the property that $X \leq_p Y$, then $Y \in \mathcal{NP}$ -complete.*

Proof. Let Z be any problem in $\mathcal{NP}$. We have $Z \leq_p X$ by Definition 10, and $X \leq_p Y$ by assumption. By transitivity of $\leq_p$ (reductions), it follows that $Z \leq_p Y$. $\square$

Theorem 3 ($\text{SAT} \equiv_p \text{CircuitSAT}$). *SAT belongs to $\mathcal{NP}$ -complete.*

Proof. Clearly SAT is in $\mathcal{NP}$, since we can verify in polynomial time that a proposed truth assignment satisfies the given set of clauses. We will prove that SAT belongs to $\mathcal{NP}$ -complete via the reduction $\text{CircuitSAT} \leq_p \text{SAT}$. There is a straightforward transformation from CircuitSAT to SAT by CNF encoding (Section 1.2.3) in polynomial time. Conjoining the clauses from all the gates with an additional clause constraining the circuit’s output variable to be 1 completes the reduction; a truth assignment of the variables satisfying all of the constraints exists if and only if the original circuit is satisfiable, and any **yes**-solution to CNF is a solution to the original problem of finding inputs that make the circuit output 1. The converse, that $\text{SAT} \leq_p \text{CircuitSAT}$, is even easier—we can rewrite the CNF as a circuit and solve that. $\square$

Corollary 3 ($3\text{-SAT} \equiv_p \text{SAT}$). *3-SAT belongs to $\mathcal{NP}$ -complete.*

Proof. In analogy with the proof of Theorem 3, there is a straightforward transformation from SAT to 3-SAT (Section 1.2.3) in polynomial time. $\square$

We now list the basic genres of $\mathcal{NP}$ -complete problems and paradigmatic examples, to feeling for the breadth of this phenomena. Additionally, hundreds more can be found in [30].

- *Packing + covering problems:* SET/VECTEX-COVER, INDEPENDENT-SET.
- *Constraint satisfaction problems:* CIRCUITSAT, SAT, 3-SAT.

³ In this section, CircuitSAT (resp. SAT, 3-SAT) consists of all **yes**-instances which correspond **satisfiable** propositional logic formulae; while CircuitUNSAT (resp. UNSAT, 3-UNSAT) consists of all **no**-instances which correspond **unsatisfiable** propositional logic formulae.

- *Sequencing problems*: HAM-CYCLE, KNAPSACK, TSP.
- *Partitioning problems*: 3-COLOR, 3D-MATCHING.
- *Numerical problems*: SUBSET-SUM, NUMB-PARTITIONING.

Corollary 4. $\mathcal{P}_{\text{yes}} = \mathcal{NP}$ iff $\text{SAT} \in \mathcal{P}_{\text{yes}}$

To summarize, we are achieving a mathematics’ holy grails of understanding structure, namely *necessary* and *sufficient* conditions, sometimes phrased as a *duality theorem*. SAT captures the difficulty of the whole class $\mathcal{NP}$. In particular, the millennium prize problem— $\mathcal{P}$ vs. $\mathcal{NP}$ ⁴ can now be phrased as a question about the complexity of *one* problem, instead of infinitely many.

1.4 Organization

The organization of this dissertation is as follows.

Chapter 2 will be concerned with sound and complete SAT algorithms that are guaranteed to terminate with a correct decision on the satisfiability/unsatisfiability of a given CNF formula. We will start by describing the classical Davis-Putnam-Logemann-Loveland (DPLL) algorithm. Afterwards, an interplay between search and inference will be explicated, and we will summarize the organization of modern conflict-driven clause learning (CDCL) solver.

Chapter 3 will pitch our new thought of two efficient implementations for improving the current state-of-the-art CDCL solver. One focus on the special strategy of pure literal elimination which is regarded as another deduction engine of SAT algorithms, besides unit propagation. The other enhances the management of learning and deleting conflict-driven clauses by a breadth-extended approach. Both of these two implementation were introduced into our submission solvers that participated in the recent SAT competitions.

In Chapter 4, we will discuss the encoding of pseudo-Boolean (PB) constraints in the context of a linear search satisfiability-based solver for maximum satisfiability (MaxSAT). We will present several CNF encodings of PB constraints for weighted partial MaxSAT and provide detailed proofs of the correctness for all our encodings. An n -level modulo totalizer, which is the newest of them, applies multiple modular arithmetic in a mixed radix base and only generates auxiliary variables for each unique combination of weights. Significantly, this encoding always produces a polynomial-size CNF in the number of input variables.

In Chapter 5, we will study the coalition structure generation (CSG) problem for partition function games (PFGs) which are coalitional games with externalities. In PFGs, each value of a coalition depends on how the other agents are partitioned. We propose and benchmark two methods to solve CSG problem for PFGs, concisely represented by partition decision trees (PDTs). One is a depth-first branch-and-bound (DFBnB) algorithm for modified PDTs, called the MPDT algorithm, and the other is MaxSAT encoding and leaves the rest manipulation to MaxSAT solver.

Finally, Chapter 6 will contain a summary of this dissertation and discussion of some future research directions that may be worth exploring.

⁴ <http://www.claymath.org/millennium-problems/p-vs-np-problem>

Chapter 2

SAT Algorithms

This chapter is concerned with *sound*¹ and complete algorithms for testing satisfiability, i.e., algorithms that are guaranteed to terminate with a correct decision on the satisfiability/unsatisfiability of given CNF $\mathcal{F}$. One can distinguish between a few approaches on which complete satisfiability algorithms have been based.

The first approach is based on *existential quantification*, where one equisatisfiably eliminates variables from the $\mathcal{F}$ which is applied to having come from a prenex form with its clauses grounded in *resolution* inference rule. The Davis-Putnam (DP) algorithm [25], reduced the size of the search space considerably by this approach. This was done by repeatedly choosing a target variable x_i still appearing in the expression, applying all possible ground resolutions on x_i , then eliminating all remaining clauses still containing x_i or $\neg x_i$. When all variables have been eliminated, the satisfiability test is then reduced into a simple test on a trivial CNF ($\mathcal{F} = \emptyset$).

The second approach is based on *systematic search* in the space of truth assignments. Loveland and Logemann attempted to implement DP but they found that ground resolution used too much random access memory (RAM), which was quite limited in those days. So they changed the way variables are eliminated by employing the *splitting rule*: recursively assigning values 0 and 1 to a variable and solving both resulting subproblems [24]. Their algorithm is, of course, the familiar Davis-Putnam-Logemann-Loveland (DPLL), which is marked by its modest space requirements.

The last approach is based on combining search and inference, leading to algorithms that currently underly most modern complete SAT solvers—*conflict-driven clause learning* (CDCL). Besides using DPLL, implementing a state-of-the-art CDCL SAT solver involves a number of additional key techniques:

- Learning new clauses from conflicts during backtrack search [65].
- Exploiting structure of conflicts during clause learning [65].
- Using lazy data structures for the representation of formulae [55].
- Branching heuristics must have low computational overhead, and must receive feedback from backtrack search [55].

¹ A valid argument whose premises are tautology is said to be sound.

- Periodically restarting backtrack search [33].
- Deletion strategies and policies for learnt clauses [32].

We start in the next section by describing the classical DPLL algorithm. Afterwards, Section 2.2 explicates an interplay between search and inference and Section 2.3 summarizes the organization of modern CDCL solvers.

2.1 The DPLL Algorithm

The DPLL algorithm [24] is based on systematic search in the space of truth assignments which was developed in response to the challenges posed by the space requirements of the DP algorithm [25].

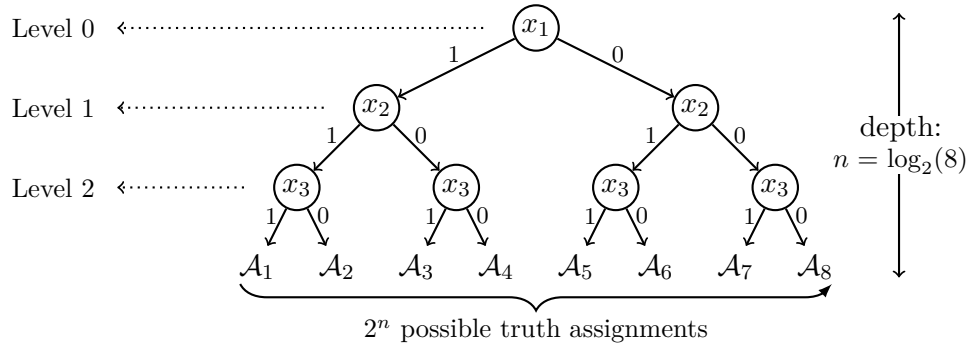


Figure 2.1. A search tree of enumerating all truth assignments over Boolean variables x_1 , x_2 and x_3 in decision variable order $\overrightarrow{x_1 x_2 x_3}$ which corresponds the decision levels from top to bottom in the leftmost of figure.

Figure 2.1 depicts a search tree that enumerates the truth assignments over Boolean variables $\mathcal{X} = \{x_1, x_2, x_3\}$ in *decision variable order* $\overrightarrow{x_1 x_2 x_3}$. *Decision levels* (starting from level 0) of each assignment reflects this order. The observation about this tree is that it essentially is an n -depth *complete binary tree*, where $n = |\mathcal{X}|$, i.e., the number of variables, and its leaves are in one-to-one correspondence with the truth assignments under consideration. Hence, testing satisfiability can be viewed as a process of searching for a leaf node that satisfies the given CNF. Obviously, a brute-force search for all possible assignments is in $O(2^n)$. The *decision level* associated with variables used for branching steps (i.e., decision assignments) is specified by the search process, and denotes the current depth of the *decision stack*.

A DPLL explores the tree using depth-first-search algorithm, which incorporates *unit propagation* (UP), also called *unit resolution*. Given our notational conventions in Definition 1, a CNF $\mathcal{F}$ is **sat** if $\mathcal{F} = \emptyset$. Moreover, $\mathcal{F}$ is **unsat** if $\square \in \mathcal{F}$. These two properties correspond to common boundary conditions that arise in recursive algorithms on CNF. In general, a CNF $\mathcal{F}$ is said to *imply* another CNF Γ , denoted $\mathcal{F} \models \Gamma$, iff every assignment that satisfies $\mathcal{F}$ also satisfies Γ . Note that if a clause C_i is a subset of another clause C_j , $C_i \subseteq C_j$, then $C_i \models C_j$. We say in this case that clause C_i *subsumes* clause C_j , as there is no need to have clause C_j in a CNF that also includes clause C_i . One important fact based on this discussion is that $\mathcal{F}$ is intrinsically **unsat** if $\mathcal{F} \models \square$.

2.1.1 Unit propagation

Example 6. Consider the following CNF:

$$\mathcal{F} = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3)$$

Consider also the node at Level 1 on the left branch as shown in Figure 2.1, which results from assignment $\mathcal{A}(x_1) = 1$, and we have:

$$\mathcal{A}(\mathcal{F}) = (\neg x_3) \wedge (\neg x_2 \vee x_3) \quad (2.1)$$

Currently, $\mathcal{F}$ is neither empty ($\emptyset$), nor includes the empty clause ($\square$). Therefore, we cannot yet declare early success (**sat**) or early failure (**unsat**), which is why a brute-force algorithm continues searching below Level 1.

However, that by employing UP, we can indeed declare success and end the search at this node. Firstly, we close $\mathcal{F}$ under UP and collect all *unit clauses*² in $\mathcal{F}$. Then we assume that literals are assigned to 1 to satisfy these unit clauses. Lastly, we simplify $\mathcal{F}$ given these assignments and check for either that all clauses are subsumed or that the empty clause is derived.

To incorporate UP into DPLL algorithm, we introduce a function UNITPROPAGATE, which applies to a CNF $\mathcal{F}$ and returns two results:

- $\mathcal{L}$: a set of literals that were either present as unit clauses in $\mathcal{F}$, or were derived from $\mathcal{F}$ by UP.
- Γ : a new CNF which results from assignments for $\mathcal{F}$ on literals $\mathcal{L}$ conditioning to assign all of them to 1. Notationally, $\Gamma = \mathcal{A}(\mathcal{F}) \mid \forall l (l \in \mathcal{L}) (\mathcal{A}(l) = 1)$.

Continuously consider Formula 2.1, we have $\mathcal{L} = \{\neg x_3, \neg x_2\}$ and $\Gamma = \emptyset$.

UP is a very important component of search-based SAT solving algorithms because of its efficiency that we can apply all possible UP steps in time linear in the size of given CNF. DPLL is a refinement on depth-first-search which incorporates UP as given in Algorithm 1. Beyond applying UP on Line 1, we have two additional changes to ordered brute-force search as shown in Figure 2.1. First of all, we no longer assume that variables are examined in the given decision variable order as we go down the search tree. Secondly, we no longer assume that a variable is assigned to 1 first, and then to 0; see Line 5. The particular choice of a literal l s.t. $l \in \Gamma$ can have a dramatic impact on the running time of DPLL. Many heuristics have been proposed for making this choice. Among them, a quite effective heuristic, well-known as *variable state independent decaying sum* (VSIDS) attempts to score each variable and select the *variable polarity* (corresponding to literal and its negation) with the highest score to determine where to branch [55].

2.1.2 Chronological backtracking

DPLL (Algorithm 1) is based on *chronological backtracking*. That is, if we try both values of a variable at Level t , and each leads to a contradiction (*conditional unsat*),

² If a clause only contains a single literal, then it is called a unit clause.

Algorithm 1: DPLL(CNF $\mathcal{F}$): returns a set of literals or **unsat**.

```

1  $(\mathcal{L}, \Gamma) \leftarrow \text{UNITPROPAGATE}(\mathcal{F});$ 
2 if  $\Gamma = \emptyset$  then return  $\mathcal{L}$ ;
3 else if  $\square \in \Gamma$  then return unsat;
4 else
5   CHOOSE a literal  $l$  in  $\Gamma$ ;
6   if  $\mathcal{I} \leftarrow \text{DPLL}(\mathcal{A}(\Gamma) \mid \mathcal{A}(l) = 1) \neq \text{unsat}$  then return  $\mathcal{I} \cup \mathcal{L} \cup l$ ;
7   else if  $\mathcal{I} \leftarrow \text{DPLL}(\mathcal{A}(\Gamma) \mid \mathcal{A}(\neg l) = 1) \neq \text{unsat}$  then return  $\mathcal{I} \cup \mathcal{L} \cup \neg l$ ;
8   else return unsat;
9 end
```

we move to Level $t - 1$, undoing all intermediate assignments in the process, and try another value at that level (if one remains to be tried). If there are no other values to try at Level $t - 1$, we move to Level $t - 2$, and so on. If we move all the way to Level 0, and each value there leads to a contradiction, we know that the CNF is inconsistent (*ultimate unsat*).

The process of moving from the current level to a lower level is known as *backtracking*. Moreover, if backtracking to Level t is done only after having tried both values at Level $t + 1$, then it is called *chronological backtracking*, which is the kind of backtracking used by DPLL. The problem is that it does not take into account the information of the contradiction that triggers the backtrack.

Example 7. Consider a concrete CNF as follows:

$$\mathcal{F} = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6 \text{ where } \begin{cases} C_1 = (x_1 \vee x_2) \\ C_2 = (x_2 \vee x_3) \\ C_3 = (\neg x_1 \vee \neg x_4 \vee x_5) \\ C_4 = (\neg x_1 \vee x_4 \vee x_6) \\ C_5 = (\neg x_1 \vee \neg x_5 \vee x_6) \\ C_6 = (\neg x_1 \vee x_4 \vee \neg x_6) \\ C_7 = (\neg x_1 \vee \neg x_5 \vee \neg x_6) \end{cases} \quad (2.2)$$

Figure 2.2 depicts the *termination tree*³ for DPLL on $\mathcal{F}$, which is assumed that it starts first by $\mathcal{A}(x_1) = 1$ and explore the subtree below that node, only to find that all branches lead to contradictions. This basically means that $\mathcal{A}(\mathcal{F}) \mid \mathcal{A}(x_1) = 1$ inevitably is inconsistent, because an **unsat**-core (highlighted in red) is included in it. Note, however, that DPLL is unable to discover this fact in the knowledge until it first backtrack to Level 0.

The key point here is that all these contradictions that are discovered deep in the search tree are actually caused by having $\mathcal{A}(x_1) = 1$ at Level 0. That is, the assignment for x_2 at Level 1 is irrelevant here. However, chronological backtracking is not aware of this fact. As a result, it tries different values of variable x_2 , hoping to fix the conflict. As shown in Figure 2.2, DPLL with chronological backtracking

³ Termination tree is the subset of search tree that is actually explored during search.

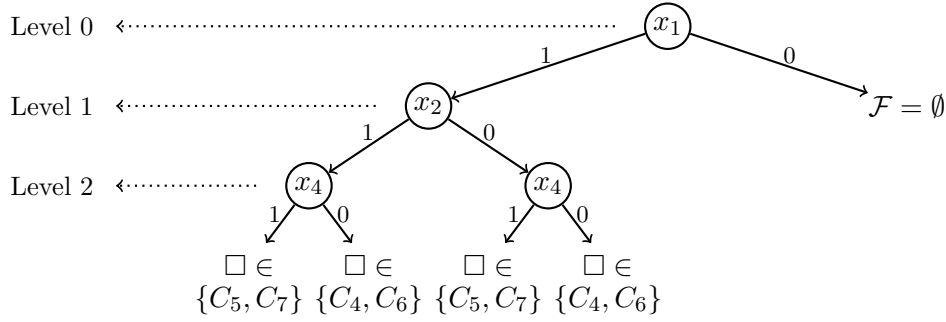


Figure 2.2. A termination tree. Assignments shown next to nodes are derived using UP.

encounters 4 contradictions before realizing that $\mathcal{A}(x_1) = 1$ was a wrong decision. In general, there can be many irrelevant variables between the level of conflict and the real cause of the conflict. Chronological backtracking may lead the solver to repeat the same conflict over and over again in different assignment for irrelevant variables.

2.2 Search and Inference

In modern SAT solvers, the utilization of *non-chronological backtracking* is tightly coupled with the derivation of conflict-driven clauses. Now we explicate the method that uses to derive conflict-driven clauses from contradictions and the specific way of search that performs non-chronological backtracking. The combination of these techniques makes sure that UP is empowered every time a conflict arises and that the solver will not repeat any mistake. The identification of conflict-driven clauses is done through a process known as *conflict analysis*, which analyzes a trace of UP.

2.2.1 Implication graph

When the UP deduces an assignment (i.e., $\mathcal{A}(l) = 1$), it is due to a clause, which was/became unit. This clause is called the *reason* or *antecedent* of the assignment. The antecedents for literal l is denoted as $\text{ANTE}(l)$. An assignment has no antecedent if it is due to a decision. The antecedent relations between literals and clauses can be expressed in a form of an oriented structure, called the *implication graph* [65].

Definition 11 (Implication graph). An implication graph is a directed acyclic graph G which is a tuple $G = \langle f, V, E \rangle$, where V (resp. E) is a set of assignments, each of whose element is labeled by occurring decision level (resp. a set of edges, each of whose element is labeled by a clause including $\text{ANTE}(V)$) and $f : \text{ANTE}(V) \times V \mapsto E$.

An implication graph is a bookkeeping device that allows us to record dependencies among assignments as they are established by UP. Each node (vertex) in implication graph has the form $\mathcal{A}(x_i) = 0/1 @ t$, which means that variable x_i has been assigned to 0/1 at level t . The edges which are marked by the same reason clause jointly imply an assignment by UP. For knowledge base in Example 7, we have the implication graph of Figure 2.3 before its first chronological backtracking. E.g., implying assignment $\mathcal{A}(x_5) = 1$ at Level 2 by using C_3 , $\mathcal{A}(x_1) = 1$ at Level 0 and $\mathcal{A}(x_4) = 1$ at Level 2. Implying will not stop until a conflict occurs. E.g., in

Figure 2.3, due to $\text{ANTE}(x_6)$ in C_5 , UP derives $\mathcal{A}(x_6) = 1$ and $C_7 = \square$, and due to $\text{ANTE}(\neg x_6)$ in C_7 , UP derives $\mathcal{A}(x_6) = 0$ and $C_5 = \square$, which is a contradiction.

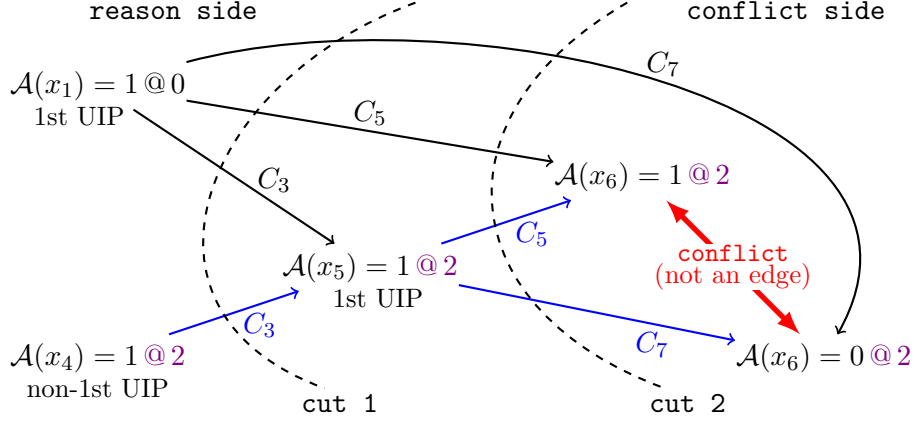


Figure 2.3. Two cuts in an implication graph, and an example of a UIP.

2.2.2 Deriving a conflict-driven clause

The conflict-driven clause can be derived by partitioning the implication graph into two parts. The partition has all the decision assignments on one side (*reason side*) and the conflicting assignments by UP on the other side (*conflict side*). The described partitioning is called a *cut*. Nodes on the reason side contribute to the conflict-driven clause if they have at least one edge to the conflict side. We call these nodes as *conflict sets*. Intuitively, a conflict set contains assignments that are sufficient to cause the conflict. Figure 2.3 depicts cut 1 and cut 2 leading to conflict sets $\{\mathcal{A}(x_1) = 1, \mathcal{A}(x_4) = 1\}$ and $\{\mathcal{A}(x_1) = 1, \mathcal{A}(x_4) = 1, \mathcal{A}(x_5) = 1\}$, respectively. Once a conflict set is identified, a conflict-driven clause can be obtained by simply negating the assignments in the set. E.g., if the conflict set is $\{\mathcal{A}(a) = 1, \mathcal{A}(b) = 0, \mathcal{A}(c) = 0\}$, then the conflict-driven clause derived is $(\neg a \vee b \vee c)$. This clause represents the previously-implicit constraint that the assignments in the conflict set cannot be made simultaneously ($\mathcal{A}(a) = 1, \mathcal{A}(b) = 0, \mathcal{A}(c) = 0$ will now violate this new clause).

Given the multiplicity of conflict sets, one is interested in choosing the most useful one. In practice, virtually all modern SAT solvers insist on selecting cuts that involve exactly one variable assigned at the level of conflict for a reason that we become apparent in the next subsection.

2.2.3 Asserting clauses

A conflict-driven clause is *asserting* if it contains exactly one variable that has been assigned at the decision level where the contradiction is found. The main motivation for identifying asserting clauses is to reduce the size of learnt (derived) clauses. It is useful and even necessary for completeness of the SAT algorithms that generating asserting conflict-driven clauses, because a smaller clause potentially allows the solver to prune more search space. To find cuts, which lead to asserting clauses, we need to locate *unique implication points* (UIPs). In particular, a UIP

of a decision Level t in an implication graph is a dominator at Level t , that all oriented paths from the decision node at Level t to one of nodes causing conflict go through this dominator. Notice that the decision node itself is always a UIP. There are sophisticated algorithms for computing dominators in directed acyclic graphs [68]. For the case of the implication graph, UIPs can be identified in linear time [64], and so do not add significant overhead to the clause learning procedure. In Figure 2.3, the assignments $\mathcal{A}(x_4) = 1$ and $\mathcal{A}(x_5) = 1$ would be UIPs of Level 2 as every path from decision node $\mathcal{A}(x_4) = 1$ at Level 2 to the contradiction.

Note that there may be more than one UIP for a given level and contradiction. In such a case, we order the UIPs according to their distance to the node causing conflict. The *first UIP* is the one closest to the contradiction. Even though there are many possible ways of generating an asserting conflict-driven clause from different UIPs of distinct decision level, the preferable one is that only consist of the first UIP of the conflict level and all decision nodes of other levels contained in corresponding conflict set. The reason is that the size of such an asserting conflict-driven clause tend to be the smallest, and besides, in UP after learning this clause and backtracking, solver will not repeat unnecessary implications for those literals in non-first UIP of the last conflict level until discovering the next conflict. This scheme of deriving asserting clauses is known as *1UIP scheme*. Considering Figure 2.3, the conflict set of cut 2 is $\{\mathcal{A}(x_1) = 1, \mathcal{A}(x_4) = 1, \mathcal{A}(x_5) = 1\}$, leading to an asserting conflict-driven clause $(\neg x_1 \vee \neg x_5)$, because $\mathcal{A}(x_5) = 1$ corresponds to the first UIP of Level 2 (i.e., the conflict level), and $\mathcal{A}(x_1) = 1$ corresponds the decision node of other level (in this case, there exists only Level 0) contained in this conflict set.

Notice that there is a special case of deriving an asserting conflict-driven clause for the following condition. If the conflict set of a cut contains only assignments from one level, in other words, the asserting clause should have only one literal. Whenever this happens, we backtrack to Level 0, and then learn an asserting clause which is obtained by negating the assignment of decision node at conflict level. The reason is simple. That is, because this decision is unconditional wrong. Thus, before any decision, this wrong becomes unit and allows UP to force a reverse implication.

2.2.4 Backtracking Scheme

After the process of adding an asserting conflict-driven clause to the CNF, which is known as *clause learning*, a key question in this regard is: Where (decision level) to jump? To continue solving Formula 2.2 based on identifying the conflict-driven clause $(\neg x_1 \vee \neg x_5)$, we add it into CNF (i.e., $\mathcal{F} \wedge (\neg x_1 \vee \neg x_5)$) and then have to at least undo the decision level of conflict (Level 2), since the contradiction was discovered after assignment $\mathcal{A}(x_4) = 1$. However, in modern SAT solvers, whenever a conflict is encountered, the level for backtracking is determined from the learnt clause. It is reasonable, since the learnt clause contains information about the conflict. We will undo all decisions and of course, their implications made after the *assertion level*, which is the second deepest level in a conflict-driven clause. For example, in the clause $(\neg x_1 \vee \neg x_5)$, x_1 was assigned at Level 0 and x_5 was assigned at Level 2. Hence, the assertion level is 0 in this case. Notice that if the learnt asserting clause just have a single literal, which is the special case mentioned in the last subsection, its assertion level is then -1 by definition. There will never exist any decision

assignment, but implications by UP at Level -1 . Observe that when each learnt clause is created it has all literals but one assigned value 0, so it is unit (or *assertive*). Moreover, each learnt clause will remain unit until the search procedure backtracks to the highest decision level of its other literals. This form of clause learning, and associated backtracking procedure, is referred to as *first UIP clause learning*.

2.3 Organization of CDCL Solver

Algorithm 2 shows the standard organization of a CDCL SAT solver, which essentially follows the organization of DPLL. With respect to DPLL, the main differences are the call to function CONFLICTANALYSIS at each time a conflict is identified, and it returns a learnt clause and a decision level where should backtrack; see Line 13. Moreover, the conflict-driven DPLL procedure allows for backtracking non-chronologically.

Algorithm 2: CDCL(CNF $\mathcal{F}$): returns **sat** or **unsat**.

```

Initialize:  $\mathcal{A}$ , level  $\leftarrow -1$ 
1   $(\mathcal{L}, \Gamma) \leftarrow \text{UNITPROPAGATE}(\mathcal{F});$ 
2  if  $\square \in \Gamma$  then return unsat;
3  else
4    forall the  $l$  ( $l \in \mathcal{L}$ ) do  $\mathcal{A}(l) \leftarrow 1$ ;
5  end
6  level  $\leftarrow 0$ ;
7  while true do
8     $l \leftarrow \text{DECIDE}(\Gamma, \mathcal{A});$ 
9    if  $l = \text{null}$  then return sat;
10   level  $\leftarrow \text{level} + 1$ ,  $\mathcal{A}(l) \leftarrow 1$ ;
11    $(\mathcal{R}, \Delta) \leftarrow \text{UNITPROPAGATE}(\mathcal{A}(\Gamma) \mid \mathcal{A}(l) = 1);$ 
12   if  $\square \in \Delta$  then
13     (learnt, backtrack)  $\leftarrow \text{CONFLICTANALYSIS}(\Delta, \mathcal{A});$ 
14     if level =  $-1$  then return unsat;
15      $\Gamma \leftarrow \Gamma \wedge \text{learnt};$ 
16      $\mathcal{A} \leftarrow \text{UNDO}(\mathcal{A}, \text{backtrack});$ 
17     level  $\leftarrow \text{backtrack};$ 
18   else
19     forall the  $l$  ( $l \in \mathcal{R}$ ) do  $\mathcal{A}(l) \leftarrow 1$ ;
20   end
21 end

```

From Line 7 to 9, after entering an infinite cycle, CDCL makes a decision to select a branching literal. Notice that if no branching literal can be decided, then all must be assigned. In this case, we have satisfying assignments for every variable and solving is finished. For another termination condition shown in Line 14, a conflict at Level -1 means that there is a variable, which must be 0 and 1 at the same time, because the learnt clauses include two asserting clauses which correspond to a pair of complementary literals. Undoubtedly, the answer is unsatisfiable.

The typical CDCL algorithm presented does not account for a few recent techniques that have been shown to be effective in practice, namely search restarts [33], implementation of clause deletion strategies and policies [32] and lazy data structures for the representation of formulae [55]. Search restarts cause the algorithm to restart itself, but already learnt clauses are kept. Clause deletion strategies and policies are used to decide learnt clauses that can be deleted, which allows the memory usage of the SAT solver to be kept under control. Lazy data structures also plays an important role for implementation issues, including the design of suitable data structures for storing clauses, assignments of variables, and the orders of decisions and implications, which have a significant impact on the run time performance of SAT solvers. Clause deletion and lazy data structures are tightly related to our original works will be discussed in the following chapter.

There is one further characterization. DPLL is a sound and complete algorithm for SAT [25, 24]. CDCL solvers implement DPLL, but can learn new clauses and backtrack non-chronologically. Clause learning with conflict analysis identifies new clauses using the *resolution* operation. Hence each learnt clause can be inferred from the original clauses and other learnt clauses by a sequence of resolution steps. If C_l is the new learnt clause, then $\mathcal{F}$ is satisfiable if and only if $\mathcal{F} \wedge C_l$ is also satisfiable. In addition, the modified backtracking step also does not affect soundness or completeness, since backtracking information is obtained from each new learnt clause. Proofs of soundness and completeness for different variations of CDCL SAT solvers can be found in [62, 78].

Chapter 3

Efficient Implementations

The efficiency of modern SAT solvers relies heavily on various features that have been developed, analyzed, and tested over the last decade. The *International SAT Solver Competition*¹ is today an established series of competitive events aiming at objectively evaluating the progress in state-of-the-art SAT solving techniques [40]. Over these years, the competitions have significantly contributed to the fast progress in SAT solver technology that has made SAT become a practical success story of computer science. In this chapter, we report some efficient implementations about pure literal elimination, clauses learning and deletion, which are implemented in our submission solvers that participated in the recent SAT competitions.

Representation in programming

Somehow the SAT instance must be represented by internal data structures, as must any derived information. We will use, but not specify an implementation of the abstract data type `vec<T>` an extensible vector of type `T`. The interfaces of this abstract data type is presented in Figure 3.1.

```

class vec<T> - Public interface
  Constructor:
    VEC(int size, T pad)
  - Size operators:
    int    SIZE    ()
  - Stack interface:
    void   PUSH    (T elem)
  - Vector interface:
    T      OP [ ]  (int index)

```

Figure 3.1. A basic abstract data type `vec<T>` used throughout the code.

For example, a CNF $\mathcal{F}$ can be considered as a vector of clauses, i.e., `vec<cla>` $\mathcal{F}$; a clause C_i can be considered as a vector of literals, i.e., `vec<lit>` C_i . Type `lit` is a data structure which includes the index (`int`) of a literal on the array of all constructed literals, i.e., `int lit.x`. We can also get the `lit` through whose index

¹ <http://www.satcompetition.org>

Table 3.1. The correspondence of indexes between literal and variable.

Variable	x_0		x_1		x_2		$\dots$	x_n	
Polarities	x_0	$\neg x_0$	x_1	$\neg x_1$	x_2	$\neg x_2$	$\dots$	x_n	$\neg x_n$
int lit. x	0	1	2	3	4	5	$\dots$	$2n$	$2n + 1$

by `lit GETLIT(int lit. $x$ )` method. For assignment state, we use `assigns` as a vector of current assignments indexed on entire variables, i.e., `vec<lbool> assigns`, where `lbool` is a type of Boolean domain with three constants 0, 1, and `*` (standing for *unassigned*). We also use `trail` as a vector of literals which have been assigned to 1 in chronological order, i.e., `vec<lit> trail`. The correspondence of indexes between literal and variable is shown in Table 3.1.

3.1 Pure Literal Elimination

Besides UP, there are other rules that can be incorporated into a deduction engine. This section will briefly review one of them. It should be pointed out that though many of the deduction mechanisms have been shown to work on certain classes of SAT instances, unlike UP, none of them seems to work without deteriorating the overall performance of the SAT solver for general SAT instances.

One of the most widely known rules for deduction is the *pure literal rule* [25], which was introduced into DP algorithm for variables elimination. A variable is said to be *pure* if it only occurs in a single polarity in all the unresolved clauses. The pure literal rule states that if a given CNF in its partial assignment includes any pure variables in the unresolved clauses, then each of these pure variables can be assigned with a value such that the literal of the variable in that polarity evaluates to 1. Hence, this amounts to the same as removing all clauses that include any pure variables without affecting satisfiability of the CNF.

Since the pure literal rule may lead to other literals be coming pure, the process needs to be iterated to obtain an equisatisfiable CNF without any pure literal. This process is known as *pure literal elimination* (PLE) [41]. In practice, it is expensive to detect whether a variable satisfies the pure literal rule during the actual solving process. Therefore most SAT solvers do not use PLE as a deduction process by default, while there are few modern solvers still implement PLE only in their preprocessing polarity.

3.1.1 Priorities of variable assignment

In the major solving procedure of CDCL solvers, variable assignment essentially happens in two different situations: the one is consists of the identification of unit clauses and the creation of the associated implications which carries out UP; the other one is decision assignment which picks an unassigned literal by a decision strategy, such as VSIDS heuristics [55]. With defining a function `PRIORITY` as the priority evaluation of variable assignment, we can obviously know that `PRIORITY(UP) > PRIORITY(Decision)`. Here we reconsider that PLE as the third situation of variable assignment which has a higher priority than decision assignment, as well as UP, i.e., `PRIORITY(PLE) > PRIORITY(Decision)`. PLE can also be a deduction engine which

evaluates a variable into the reasonable value. However, most CDCL solvers which have undergone many refinements are based on UP as a core deduction process. In order to conveniently implement the twin-engined SAT solver, in this section we think the relationship of the priorities among UP, PLE and decision assignment as following inequation:

$$\text{PRIORITY}(\text{UP}) \geq \text{PRIORITY}(\text{PLE}) > \text{PRIORITY}(\text{Decision})$$

Example 8. Consider the following CNF with convention (representation in programming) of the indexes starting from 0 in vectors:

$$\mathcal{F} = C_0 \wedge C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \text{ where } \begin{cases} C_0 = (x_0) \\ C_1 = (x_0 \vee x_1 \vee \neg x_3) \\ C_2 = (\neg x_0 \vee \textcolor{red}{x_3} \vee \textcolor{red}{x_4}) \\ C_3 = (\neg \textcolor{red}{x_1} \vee x_2) \\ C_4 = (\neg x_2 \vee \textcolor{red}{x_3}) \\ C_5 = (\textcolor{red}{x_1} \vee \neg x_4) \end{cases} \quad (3.1)$$

After assigning x_0 to 1 by UP, $\mathcal{F}$ is eliminated and the part remained (highlighted in red) includes a pure literal x_3 . PLE obeys the pure literal rule which leads to assignment $\mathcal{A}(x_3) = 1$. Then only C_3 and C_5 remains in $\mathcal{F}$ which also include pure literals x_2 and $\neg x_4$, respectively. Eventually, without any decision assignment, the twin-engined SAT solver will derive $\mathcal{F} = \emptyset$ from $\text{trail} = \langle 0, 6, 4, 9 \rangle$ which identically means the orderly assignments $\mathcal{A}(x_0) = 1$, $\mathcal{A}(x_3) = 1$, $\mathcal{A}(x_2) = 1$ and $\mathcal{A}(x_4) = 0$.

3.1.2 Pure literal detection

It is necessary to obtain the *positions* of each literal if you want to know whether there exists the pure one(s) in current assignments. These positions indicate the occurrence of each literal in corresponding clause(s). Because clause eliminations may lead a non-pure literal to become pure. In Example 8, x_3 becomes a pure literal due to the elimination of C_1 . We use `occur`, an array of vector sizing the number of all possible literals to record the positions of each literal, i.e., `vec<cla>[2 × assigns.SIZE()] occur`, which will be initialized as shown in Initialization 3 before entire SAT solving.

Initialization 3: The occurrence of each literal in corresponding clause(s).

```

1 occur ← new vec<cla>[2 × assigns.SIZE()];
2 for i ← 0 to  $\mathcal{F}$ .SIZE() do
3   | for j ← 0 to  $\mathcal{F}[i]$ .SIZE() do occur[ $\mathcal{F}[i][j]$ ].PUSH( $C_i$ );
4 end
```

Consider Formula 3.1, the array `occur` is initialized as described in Table 3.2. Now we introduce the Algorithm 4 that can be easily implemented to detect the pure literals within linear time in number of variables. In SAT solving process, this algorithm aims at detecting all current pure literals before the timings of selecting a new decision variable. From Line 1 to 5, `eliminatedClas` marks all clauses which

Table 3.2. The initialization of the array `occur` for the instance in Formula 3.1.

Variable	Polarities	
	Positive	Negative
x_0	<code>occur</code> [0] = { C_0, C_1 }	<code>occur</code> [1] = { C_2 }
x_1	<code>occur</code> [2] = { C_1, C_5 }	<code>occur</code> [3] = { C_3 }
x_2	<code>occur</code> [4] = { C_3 }	<code>occur</code> [5] = { C_4 }
x_3	<code>occur</code> [6] = { C_2, C_4 }	<code>occur</code> [7] = { C_1 }
x_4	<code>occur</code> [8] = { C_2 }	<code>occur</code> [9] = { C_5 }

have been eliminated (satisfied) with **true**, and the rest are **false**. All unassigned variables are the candidates for pure literals and for each of them, we check whether there is no positive (resp. negative) polarity of it from Line 9 to 12 (resp. from Line 13 to 16). If a variable only occurs in a single polarity in all the unresolved clauses, then it will be pushed into `pureLits` which will be finally returned; see Lines from 17 to 23.

3.1.3 Dummy decision assignment

When some pure literals have been detected by Algorithm 4, an extended solver that carries out PLE should evaluate them to 1. In this regard, we try to avoid intervening and modifying the original CDCL procedure as much as possible. There is a simple and adequate way to manipulate such this case by setting a new decision level and doing decision assignment without any heuristics for each of detected pure literals. We call this approach *dummy decision assignment*, where the pure literal detection can be regarded as a reasonable decision variable selecting heuristics. Although dummy decision assignments will affect deriving and learning conflict-driven clauses, and non-chronological backtracking, they can always help to prune search space, since it is impossible that a conflict occurs in the decision level of any dummy decision assignment.

Nevertheless, as has been mentioned, PLE deduction engine probably deteriorate the overall performance of SAT solvers for general instances, especially burdened with those have a huge number of variables and clauses. We are well aware of that pure literal detection will keep a high execution frequency in solving process, which might reduce the efficiency of solving. Therefore, for such a twin-engined solver, it is necessary to find a balance of the execution frequency of Algorithm 4.

3.1.4 GLUCOSEPLE in SAT competition 2016

We introduce Algorithm 4 into GLUCOSE 3.0 [10, 9] and implement an approach that can effectively utilize the features of instance and current states to set an suitable limited execution frequency f of our algorithm. Assume that during a *unit time* the decision level increases by γ , without considering the circumstances under backtracking, we should ensure that the frequency is limited to γ or less. Besides, we consider the feature constant β , where $\beta = \text{assigns.SIZE}() \times \mathcal{F}.\text{SIZE}()$, as the *difficulty coefficient*, and the number α of times calling UP during a unit time as the *computation coefficient*. A dynamic adjustment approach sets an suitable limited frequency defined as $f = (\alpha/\beta) \times \gamma$. We submitted our solver GLUCOSEPLE with

Algorithm 4: Pure literal detection algorithm in solving process

```

Initialize: pureLits  $\leftarrow$  new vec<lit>,
           eliminatedClas  $\leftarrow$  new bool[ $\mathcal{F}$ .SIZE()],
           forall the  $i$  ( $0 \leq i < \mathcal{F}$ .SIZE()) do eliminatedClas[ $i$ ]  $\leftarrow$  false;
1 for  $i \leftarrow 0$  to trail.SIZE() do
2   for  $j \leftarrow 0$  to occur[trail[ $i$ ]].SIZE() do
3     | eliminatedClas[occur[trail[ $i$ ]][ $j$ ]]  $\leftarrow$  true;
4   end
5 end
6 for  $i \leftarrow 0$  to assigns.SIZE() do
7   bool noPosLit  $\leftarrow$  false, noNegLit  $\leftarrow$  false;
8   if assigns[ $i$ ] = * then  $\triangleright$  condition: variable  $x_i$  is unassigned
9     for  $j \leftarrow 0$  to occur[ $2 \times i$ ].SIZE() do
10      | if eliminatedClas[occur[ $2 \times i$ ]][ $j$ ]] = false then break;
11      | else if  $j = \text{occur}[2 \times i].\text{SIZE}() - 1$  then noPosLit  $\leftarrow$  true;
12    end
13    for  $j \leftarrow 0$  to occur[ $2 \times i + 1$ ].SIZE() do
14      | if eliminatedClas[occur[ $2 \times i + 1$ ]][ $j$ ]] = false then break;
15      | else if  $j = \text{occur}[2 \times i + 1].\text{SIZE}() - 1$  then noNegLit  $\leftarrow$  true;
16    end
17    if noPosLit  $\oplus$  noNegLit = true then  $\triangleright$  Exclusive or operator:  $\oplus$ 
18      | if noPosLit = true then pureLits.PUSH(GETLIT( $2 \times i + 1$ )));
19      | else pureLits.PUSH(GETLIT( $2 \times i$ )));
20    end
21  end
22 end
23 return pureLits;

```

a special focus on the strategy of PLE to the SAT competition 2016². In *agile track* which is the track of simple fast SAT solvers with low overhead, suitable for solving large numbers of not too hard SAT instances, GLUCOSEPLE solved 732 **sat** instances and 2367 **unsat** instances, which ranked 7th out of 26 submitted solvers.

3.2 Clauses Learning and Deletion

This section describes our another original idea for efficient SAT implementation which is about clauses learning and deletion. Such this implementation will exploit a well known lazy data structure called the *watched literal data structure*.

3.2.1 Watched literal data structure

The implemented data structures dictate the way UP and conflict analysis, and have a significant impact on the run time performance of the SAT solver which is devoted

² <https://baldur.itk.kit.edu/sat-competition-2016>

to reducing the CPU time required per each node in the search tree. Traditional SAT data structures, a.k.a. *adjacency lists data structures*, are *accurate*, meaning that is possible to know exactly the value of each literal in the clause. Conversely, the most recent SAT data structures are not accurate and therefore are called *lazy*, which are characterized by each variable keeping a reduced set of clause' references, for each of which the variable can be effectively used for declaring the clause as unit, as satisfied or as unsatisfied.

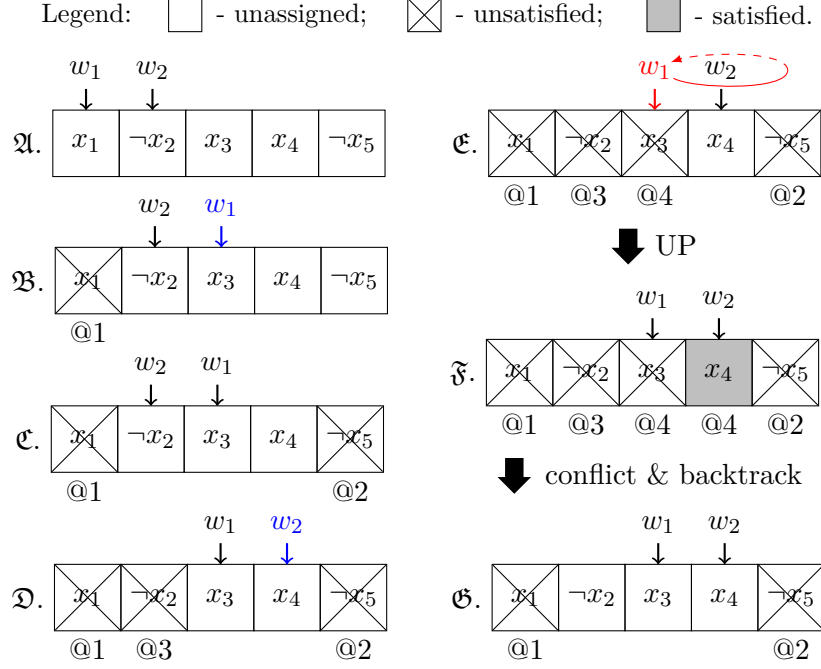


Figure 3.2. A demonstration of the watched literal data structure for clause $(x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee \neg x_5)$. After initialization (A), two watched literal references (pointers w_1 and w_2) keep updating (from B to D), until the clause is declared unit (E). Then a conflict occurs via UP (F), and no reference need to move (undo) during backtracking (G).

A famous lazy data structure proposed for SAT was the watched literal data structure, originally used in CHAFF solver [55]. As the name implies, this data structure associates the references to two watched literals with each clause which contains at least two literals. For example, we illustrate the watched literal data structure for clause $(x_1 \vee \neg x_2 \vee x_3 \vee x_4 \vee \neg x_5)$ for a *cyclic sequence* of assignments in Figure 3.2. This clause can also be represented by a cyclic array of literals. Literals have different representations depending on being unassigned, assigned value 0 (unsatisfied) or assigned value 1 (satisfied). Each assigned literal is associated with a decision level indicating the level where the literal was assigned. For sequence A, initially these two references (w_1 and w_2) point to the first (leftmost) two literals (x_1 and $\neg x_2$) respectively. The two pointed literals are said to be *watched*. From A to G, each time a watched literal changes to be unwatched if its reference becomes unsatisfied, and a new unassigned literal is searched for becoming the next watched literal. In case an unassigned literal is identified, the pointer has to move next (right or turn back), and the new reference is associated with this literal (e.g., B and

④). Notice that such these two references cannot be pointed to the same literal (i.e., $w_1 \neq w_2$). In case a satisfied literal is identified and it is not a UIP of any contradiction (i.e., no conflict occurs in current decision level), the clause is declared satisfied. In case no unassigned literal can be identified when all clause's literals are traversed, and then the clause is declared unit (e.g., ⑤), unsatisfied (e.g., ⑥) or satisfied, depending on the value of the literal watched to by another reference. Also notice that the references to the two watched literals do not need to be modified when backtracking takes place (e.g., ⑦).

In order to propagate the next unit unassigned literal in a clause, we construct a vector, called the *watch list*, whose element is the watched literal data structure orderly corresponding to each clause. In function UNITPROPAGATE, the last pushed element of `vec(lit)` trail is always specified as a watched unassigned literal in a declared unit clause by searching the watch list.

3.2.2 Breadth-extended approach

Theoretically, UP process should be iterated until no more declared unit clause. In fact, the most modern SAT solvers will stop unit propagating when a conflict occurs. It may lead to miss some opportunities to learn a “good” conflict-driven clause, e.g., a small size one, a less literal block distance (LBD) [10] one, or even a unit one. We propose a breadth-extended approach for detecting plural conflicts and learning corresponding clauses during the same decision level. This approach will not stop searching, but keep traversing the watch list. Hence, more conflicts may be detected per level. Then these conflicts will be analyzed, and corresponding conflict-driven clause will be derived. Finally, we select some good of them to be learnt clauses.

Example 9. Let's consider a situation which is depicted in Figure 3.3. In decision level 4, x_9 is firstly assigned to 0 by a decision heuristics. Then C_3 and other associated clause(s) imply $\mathcal{A}(x_3) = 0$. Next, x_4 is evaluated to 1 by UP, since C_1 is declared unit. However, C_5 and C_7 are also declared unit both of which include only one unassigned literal $\neg x_4$ (causing conflict 1 and 2). At the same time, two unit clauses emerge from C_4 and C_8 and they are declared units. In contrast, they includes a single remained unassigned literal $\neg x_6$ and x_6 respectively (causing conflict 3). More interestingly, this time, the implication $\mathcal{A}(x_8) = 1$ via the declared unit clause C_6 , and then C_2 and C_9 are chained to be declared unit, which lead to the assignments $\mathcal{A}(x_7) = 0$ and $\mathcal{A}(x_7) = 1$, respectively (causing conflict 4).

Notice that the most modern CDCL solvers only detect the conflict 1 in this example, and return the reasons of it, and then go to the learning phase. The other *detectable* conflicts (i.e., conflict 2, 3 and 4) will be brutally ignored in this decision level and there is no chance for them to generate corresponding conflict-driven clauses. The conflict 1 is the first detectable conflict, because the watched literal data structure of C_1 is the head of the watch list. Besides, in the watch list, the watched literal data structure of C_5 is in front of that of C_7 .

If we consider the source of each conflict as an inconsistency of a variable assignment, then all conflicts can be classified into two patterns as shown in Figure 3.4 which is related with Example 9, Figure 3.2 and 3.3. One is the first detected inconsistent variable assignment (i.e., x_4 in this example). Another contains all

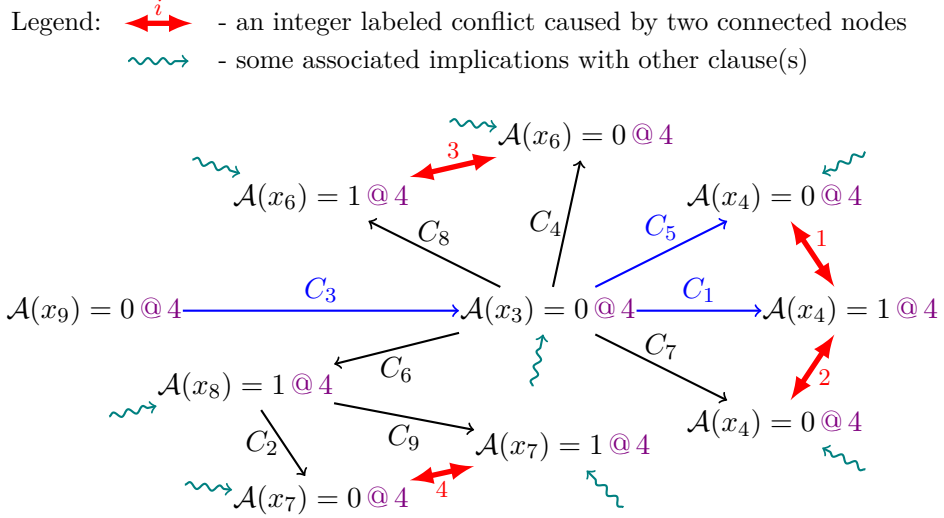


Figure 3.3. An abridged implication graph illustrating several conflicts occurring in the same decision level.

the rest which are reached by at least one more auxiliary implication. In our implementation of the breadth-extended approach, all conflicts in pattern 1 can be simply detected, but those in pattern 2 cannot be yet. Because the conflict in pattern 2 occurs after at least one more assignment via UP. Somehow the existing CDCL solver is easy to get error, if we struggle to such these assignments under duress. As a worthwhile way, we try to construct a vector of dummy assignments for this case (i.e., `vec(1bool) dummyAssigns`) and all watched literal data structures can be synchronously updated through querying both `assigns` and `dummyAssigns`. All of these modifications should be without any interference in learning phase, otherwise solving process cannot be maintained. We will test and debug our modifications, and this is also one of our future works.

At present stage, we focus on detecting all conflicts in pattern 1. Whenever a conflict occurs, we stop unit propagating (assigning), but keeps searching (traversing) watch list. After that, we store all corresponding reasons into a vector as a return of function `UNITPROPAGTE`. Then a modified `CONFLICTANALYSIS` will derive a conflict-driven clause for each passed reason in the mentioned vector. Finally, we evaluate and filter some preference among the derived conflict-driven clauses and add them into the learnt clause database.

3.2.3 Clause deletion strategies and policies

The addition of conflict-driven clauses is quite useful as it empowers UP, allowing it to detect certain contradictions that it could not detect before. The addition of conflict-driven clauses may carry a disadvantage though, as it may considerably increase the size of given CNF, possibly causing a space problem. Because unrestricted adding clause can eventually lead to the exhaustion of the available memory which is impractical in some cases. Moreover, the more clauses the CNF has, the more time it takes for the solver to perform UP. In practice, it is not uncommon for the number

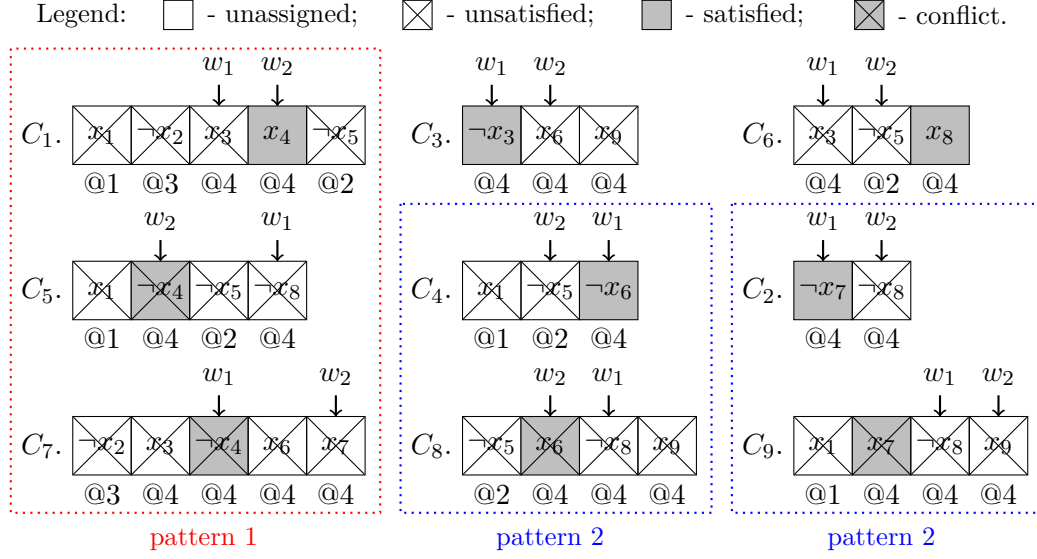


Figure 3.4. A glimpse of the watch list (only corresponding clauses from C_1 to C_9) on the brink of backtracking in Example 9, also illustrated in Figure 3.3. C_1 is the sequence of assignment which is duplicated from $\mathfrak{F}$ in Figure 3.2. The conflicts 1 and 2 (resp. conflicts 3 and 4) belong to the conflict pattern 1 (resp. pattern 2).

of added clauses to be much greater than the number of clauses in the original CNF on a difficult problem. This phenomenon may lead to additional overhead for conducting the search process and, hence, it eventually costs more than what it saves in terms of backtracks.

This issue can be dealt with in at least two ways. First, newly added conflict-driven clauses may actually subsume older conflict-driven clauses, which can be removed in this case. Second, one may decide to delete conflict-driven clauses if efficiency or space becomes an issue. In this case, the size, age, and activity of a conflict-driven clause are typically used in considering its deletion, with a preference towards deleting longer, older and less active clauses. Different conflict-driven clause deletion policies, mostly based on some heuristics, have been proposed in different SAT solvers.

3.2.4 GLUHACK in SAT competition 2018

We implement the breadth-extended approach in GLUCOSE 3.0 [10, 9] for detecting all conflicts in pattern 1 and learning the preferential derived conflict-driven clauses each of which contains only two literals or less (i.e., `learnt.SIZE() ≤ 2`). We also change the rate of learnt clause database reduction from 50% to 70%, for preserving high quality of learnt clauses. We submitted our solver GLUHACK to the SAT competition 2018³ in the *Federated Logic Conference* (FLoC) Olympic Games 2018. In *random track* which is the track of SAT solvers good on randomly generated satisfiable instances, GLUHACK solved 165 instances, which ranked second out of 10 submitted solvers.

³ <http://sat2018.forsyte.tuwien.ac.at>

Chapter 4

Pseudo-Boolean Constraint Encoding for MaxSAT

Many combinatorial problems in various fields can be translated to maximum satisfiability (MaxSAT) problems. Although the general problem is $\mathcal{NP}$ -hard, more and more practical problems may be solved due to the significant effort which has been devoted to the development of efficient solvers. The art of constraints encoding is as important as the art of devising algorithms for MaxSAT. In this chapter, we present several encoding methods of pseudo-Boolean (PB) constraint into SAT proposition, i.e., CNF formula, which are based on the idea of modular arithmetic and only generate auxiliary variables for each unique combination of weights. These techniques are efficient in encoding and solving MaxSAT problems. In particular, our solvers won the partial MaxSAT industrial category from 2010 through 2012 and ranked second place in the 2017 main weighted track of the MaxSAT evaluation. We prove the correctness and the polynomial space complexity of our encodings and also give a heuristics of the base selection for modular arithmetic. Our experimental results show that our encoding compactly encodes the constraints, and the obtained clauses are efficiently handled by a state-of-the-art SAT solver.

4.1 Maximum Satisfiability Problem

A well-known SAT problem is to determine whether there exists a model, i.e., a truth assignment that satisfies given Boolean formulas in CNF. Sometimes not all the clauses of a instance can be satisfied, and so we try to satisfy the maximum number of them, which is the idea of the MaxSAT problem, a generalization of SAT. MaxSAT can be further generalized to a weighted partial MaxSAT (WPMS) problem [5]. In this case, some clauses are declared to be *hard* that must be satisfied, and the rest are declared to be *soft* that may or may not be satisfied. We put different *weights* to each soft clause, where the weight is the penalty to falsify the clause. Hence it turns out that not all clauses are equally important. The addition of weights to clauses makes the instance *weighted*, and the separation into hard/soft clauses makes the instance *partial* [47]. Given a WPMS instance, we want to find the model that simultaneously satisfies the hard clauses and minimizes the sum of the weights of the falsified clauses.

Definition 12 (Weighted CNF). A weighted clause is a pair (C, w) , where C is a clause and w is a natural number or infinity, indicating the cost of falsifying C , which is also called its *weight*. A clause is called *hard* if the corresponding weight is infinity, and otherwise the clause is called *soft*. A WPMS instance is symbolized to a weighted CNF formula $\mathcal{F}_w$ as follows, which is a conjunction of weighted clauses:

$$\mathcal{F}_w = (C_1, w_1) \wedge \dots \wedge (C_m, w_m) \wedge (C_{m+1}, \infty) \wedge \dots \wedge (C_{m+m'}, \infty)$$

where the first m clauses are soft and the last m' clauses are hard.

For the issues of assignment for the weighted CNF, we have $\mathcal{A}((C_1, w_1) \wedge \dots \wedge (C_m, w_m)) = (\mathcal{A}(C_1), w_1) \wedge \dots \wedge (\mathcal{A}(C_m), w_m)$ and $\mathcal{A}(\emptyset) = 1$, considering $(1, w) \wedge \mathcal{F}_w = \mathcal{F}_w$ and $(0, w) \wedge \mathcal{F}_w = (\square, w) \wedge \mathcal{F}_w$ and simplifying $(C, w_1) \wedge (C, w_2) \wedge \mathcal{F}_w = (C, w_1 + w_2) \wedge \mathcal{F}_w$.

Example 10. Given $\mathcal{F}_w = (\neg x_1 \vee x_2, 3) \wedge (\neg x_2 \vee x_3, 6) \wedge (x_1 \vee \neg x_3, 2) \wedge (\neg x_1, 5)$ and $\mathcal{A} : \{x_2, x_3\} \mapsto \{0, 1\}$ such that $\mathcal{A}(x_2) = 1$ and $\mathcal{A}(x_3) = 0$, we have $\mathcal{A}(\mathcal{F}_w) = (\square, 6) \wedge (\neg x_1, 5)$.

Definition 13 (Optimal cost and assignment). Given weighted CNF $\mathcal{F}_w$ and truth assignment $\mathcal{A} : \mathcal{X} \mapsto \{0, 1\}$, the optimal cost of a formula is the minimal cost of all its total truth assignments:

$$\text{COST}(\mathcal{F}_w) = \min \left\{ \sum_{\substack{(C_i, w_i) \in \mathcal{F}_w \\ \mathcal{A}(C_i) = 0}} w_i \mid \mathcal{A} : \mathcal{X} \mapsto \{0, 1\} \right\}$$

An optimal assignment is an assignment with an optimal cost.

Example 11. Given a $\mathcal{F}_w = (\neg x_1 \vee x_2, 3) \wedge (\neg x_2 \vee x_3, 6) \wedge (x_1 \vee \neg x_3, 2) \wedge (\neg x_1, 5)$ and $\mathcal{A} : \{x_2, x_3\} \mapsto \{0, 1\}$ such that $\mathcal{A}(x_2) = 1$ and $\mathcal{A}(x_3) = 0$, we have $\text{COST}(\mathcal{F}_w) = 0$ and $\text{COST}(\mathcal{A}(\mathcal{F}_w)) = 6$.

Notice that for any weighted CNF $\mathcal{F}_w$ and total truth assignment $\mathcal{A} : \mathcal{X} \mapsto \{0, 1\}$ we have $\mathcal{A}(\mathcal{F}_w) = (\square, \text{COST}(\mathcal{A}(\mathcal{F}_w)))$. WPMS is the problem of finding an optimal assignment. If the optimal cost is infinity, which means that part of the hard clauses of formula $\mathcal{F}_w$ is unsatisfiable, then formula $\mathcal{F}_w$ is *nonoptimizable*. The weighted MaxSAT (WMS) problem is a WPMS when there are no hard clauses. In contrast, the partial MaxSAT (PMS) problem is a WPMS when the weights of the soft clauses are equal. Thus the appearance of a PMS instance is a CNF formula with internal partitions. Intimately, MaxSAT is a PMS when there are no hard clauses. Also notice that SAT is equivalent to PMS when there are no soft clauses.

4.2 QMAXSAT solver

The WPMS problem, which is a natural combinatorial optimization problem, has been scrutinized in several domains, including scheduling and timetabling problems, [3], knapsack problems [43], bioinformatics [36], FPGA routing [73], design debugging [27], and coalition structure generation [2]. Since WPMS is $\mathcal{NP}$ -hard, we cannot

solve the worst case scenario in polynomial time unless $\mathcal{P} = \mathcal{NP}$. Therefore, our aim is to come up with an efficient solver to handle real/industrial problems. The existing solvers can be classified into two approaches. One implements a branch-and-bound scheme and applies several techniques tailored to MaxSAT. Another exploits a state-of-the-art SAT solver as an inference engine. We call this the *SAT-based* approach [7]. Our solver, QMAXSAT¹ (Q-dai² MaxSAT solver) [45], follows the SAT-based approach and leaves the manipulation of the pseudo-Boolean (PB) constraint [59] to an underlying CDCL solver by encoding them into SAT.

Definition 14 (Pseudo-Boolean constraint). A pseudo-Boolean (PB) constraint (a 0-1 integer linear constraint) is an integer coefficient linear Diophantine equation or an inequation with Boolean variables, i.e., constraints of the following form:

$$\sum_{i=1}^n w_i x_i \triangleright k$$

where w_i are the coefficients of corresponding Boolean variables x_i and k is the degree, where $w_i, k \in \mathbb{Z}$, and relational operator $\triangleright \in \{=, \leq, <, \geq, >\}$. Since $-w_i \cdot x_i$ is equivalent to $w_i \cdot (\neg x_i - 1)$, these constraints can be rewritten to use only operator $\leq$ or $<$ with positive coefficients w_i . A cardinality constraint is a PB constraint where coefficients w_i equal 1.

Example 12. Given PB constraint $-147x_1 + 540\neg x_2 - 1365x_3 \geq 1477$, it can be rewritten as $147x_1 + 540x_2 + 1365x_3 \leq 2017$ or < 2018 .

Weighted CNF formula $\mathcal{F}_w = (C_1, w_1) \wedge \dots \wedge (C_m, w_m) \wedge (C_{m+1}, \infty) \wedge \dots \wedge (C_{m+m'}, \infty)$ can be solved through the resolution of a sequence of SAT instances as follows. Let $\mathcal{F}_k$ be a CNF formula that is satisfiable if and only if $\mathcal{F}_w$ has an assignment with cost less than or equal to k . One way to encode $\mathcal{F}_k$ is to extend every soft clause C_i with fresh auxiliary variable b_i , and add the conversion to the CNF of PB constraint $\sum_{i=1}^m w_i b_i \leq k$. Then

$$\mathcal{F}_k = (C_1 \vee b_1) \wedge \dots \wedge (C_m \vee b_m) \wedge C_{m+1} \wedge \dots \wedge C_{m+m'} \wedge \text{CNF}_{\text{PB}} \left(\sum_{i=1}^m w_i b_i \leq k \right)$$

If the cost of the optimal assignment to $\mathcal{F}_w$ is k_{opt} , then SAT problems $\mathcal{F}_k$, for $k \geq k_{opt}$ are satisfiable, while for $k < k_{opt}$ they are unsatisfiable. Therefore, the search of the cost of the optimal assignment to $\mathcal{F}_w$ corresponds to the precise location of this transition between satisfiable and unsatisfiable CNF formula.

Notice that k may range from 0 to $\sum_{i=1}^m w_i$ (the sum of the weights of the soft clauses). This encoding of $\mathcal{F}_k$ ensures that the set of all satisfying assignments of $\mathcal{F}_{k_{opt}}$ (with variables restricted to the variables of $\mathcal{F}_w$) is the set of optimal assignments of $\mathcal{F}_w$.

Value k_{opt} can be searched with the following different strategies: searching from $k = 0$ to k_{opt} (increasing k while $\mathcal{F}_k$ is unsatisfiable); from $k = \sum_{i=1}^m w_i$ to some value less than k_{opt} (decreasing k while $\mathcal{F}_k$ is satisfiable); or alternating unsatisfiable and

¹ <https://sites.google.com/site/qmaxsat>

² Kyushu University is called “Kyushu Daigaku” or “Q-dai” in Japanese.

satisfiable $\mathcal{F}_k$ until the algorithm converges to k_{opt} (e.g., using a binary search scheme [35]). Solvers using the first approach are sometimes called *unsatisfiability-based solvers* [29, 51, 5, 6, 14], while those using the second are called *satisfiability-based solvers*.

QMAXSAT is a satisfiability-based solver shown in Algorithm 5, which encodes $\mathcal{F}_k$ using auxiliary variables as above, exploits the information from the execution of the SAT solver for subsequent runs, and a minimal satisfying assignment can be readily found by iterative calls to the solver. First, run a SAT solver on $\mathcal{F}_k$ without a PB constraint (see Line 5) to get the truth assignment and initialize $k = \sum_{\mathcal{A}(b_i)=1} w_i$ (see Line 7), and then add PB constraint $\sum_{i=1}^m w_i b_i < k$ (see Line 8), saying that the sum of the weights of the soft clauses with auxiliary variables, which are evaluated as true, has to be less than k , and run the solver again. If $\mathcal{F}_k$ is unsatisfiable, then $k_{opt} = k$, which is the optimum solution of $\mathcal{F}_w$, or otherwise the process is repeated with the new lower cost solution. This iteration is essentially the same as solver MINISAT⁺ [26] for solving PB optimization problems. The efficiency of MaxSAT solvers depends critically on which SAT solver we use and how we encode the PB constraints. Due to Theorem 1, there is much less necessity to develop a dedicated solver that achieves the high performances of modern SAT solvers since $\mathcal{NP}$ -complete problems can be encoded in CNF formulae. The question of how best to translate PB constraints into a CNF is complicated.

Algorithm 5: QMAXSAT($\mathcal{F}_w^b$): return k_{opt}

Input : $\mathcal{F}_w^b = (C_1 \vee b_1, w_1) \wedge \dots \wedge (C_m \vee b_m, w_m) \wedge (C_{m+1}, \infty) \wedge \dots \wedge (C_{m+m'}, \infty)$;
 $\triangleright$ (1) $\mathcal{F}_w^b$: weighted CNF $\mathcal{F}_w$ augmented with auxiliary variables b_i ($1 \leq i \leq m$); (2) symbol “ ∞ ” is regarded as a finite integer, valued $1 + \sum_{i=1}^m w_i$, which is greater than sum of all soft clauses’ weights

Output : The cost of the optimal assignment to $\mathcal{F}_w$, i.e., k_{opt}

```

1 new assign  $\mathcal{A}$ ; int  $k \leftarrow \infty$ ; bool  $first \leftarrow \text{true}$ ;
2 repeat
3   if  $first$  then
4      $first \leftarrow \text{false}$ ;
5      $(\text{sat?}, \mathcal{A}) \leftarrow \text{SATSOLVER}((C_1 \vee b_1) \wedge \dots \wedge (C_m \vee b_m) \wedge C_{m+1} \wedge \dots \wedge C_{m+m'})$ ;
6   else
7      $k = \sum_{\mathcal{A}(b_i)=1} w_i$ ;
8      $(\text{sat?}, \mathcal{A}) \leftarrow \text{SATSOLVER}((C_1 \vee b_1) \wedge \dots \wedge (C_m \vee b_m) \wedge C_{m+1} \wedge \dots \wedge C_{m+m'} \wedge \text{CNF}_{\text{PB}}(\sum_{i=1}^m w_i b_i < k))$ ;
9      $\triangleright$  note that in PB constraint encoding, we only need to update the part of comparator according to current  $k$ , but keep the part of addition the same during all iterations
9   end
10 until  $\text{sat?} = \text{UNSAT}$ ;
11 return  $k$ ;
```

Koshimura et al. presented the first two versions of QMAXSAT for PMS, where one applied the totalizer (TO) encoding [11, 13] to encode *cardinality constraints*

[59] into CNF formulae, and another considered reducing the space complexity by eliminating some superfluous clauses [45]. During the first few years of this decade, QMAXSAT achieved some excellent performances in MaxSAT evaluations³, which are the annual competition of MaxSAT solvers. Nevertheless, the existing encodings suffer from memory-out and/or execution slow down caused by the huge size of the encoded formula. In 2013, Ogawa et al. proposed the first encoding based on modular arithmetic, called modulo totalizer (MTO) encoding [56], which was originally designed for PMS. QMAXSAT dramatically reduced the CNF size by utilizing MTO, and won the 2012 PMS industrial and crafted categories of the MaxSAT evaluation. Since 2015, we began focusing on efficiently solving WPMS problems and initially presented an addressing encoding called weighted totalizer (WTO) [34], based on variables partition by weighted combinations. As the same idea, generalized totalizer encoding (GTE) [42] for PB constraints was also independently presented at almost the same period. Even though WTO is exponential in the worst cases, it is competitive with other encodings. Recently, we proposed a hybrid encoding [1] which inherits modular arithmetic from MTO and distinct combinations of weights from WTO. We also enhanced the encoding by multi-level modulo arithmetic based on a mixed radix numeral system [3]. The latest version of QMAXSAT ranked second place in the main weighted track of the MaxSAT evaluation 2017.

4.3 Totalizer Based Encoding

There are many works [11, 66, 63, 18, 8] that describe techniques to encode cardinality constraints to CNF. The starting point for these works is a unary base encoding, called Totalizer (TO), proposed by Bailleux et al. [11], which uses the unary representation of integer intervals. In this section, we briefly introduce TO and its weighted extension, called WTO which is applied to encode PB constraints.

4.3.1 Totalizer

Before explaining the TO-encoding schema, we define the unary representation of an integer interval as follows:

Definition 15 (Unary representation of an integer interval). Let a , b , and u be integers that satisfy $0 \leq a \leq b \leq u$, and let $\mathcal{X} = \langle x_1 \dots x_u \rangle$ be a vector of Boolean variables. Integer interval $\tilde{x} = [a, b]$ in range $0..u$ is represented by a truth assignment $\mathcal{A} : X \mapsto \{0, 1\}$ for $X = \{x_1, \dots, x_a\} \cup \{x_{b+1}, \dots, x_u\}$ such that $\forall i (0 < i \leq a) (\mathcal{A}(x_i) = 1)$ and $\forall j (b < j \leq u) (\mathcal{A}(x_j) = 0)$. We call this assignment a unary representation of interval $\tilde{x}$ and denote it as $\tilde{x} \triangleq \mathcal{A}(X)$.

Briefly, unary representation is a vector in which only variables on the left of the vector can be assigned as 1, only variables on the right can be assigned as 0, and the variables in between are unassigned. $\mathcal{X}$ takes its values in set $\{0, 1, *\}^{|\mathcal{X}|}$ (* standing for unassigned) such that there exists two ranks, a and b ($0 \leq a \leq b \leq u$), with the following property: $x_i = 1$ if $0 < i \leq a$, $x_i = *$ if $a < i \leq b$ and $x_i = 0$ if

³ <http://www.maxsat.udl.cat>

$b < i \leq u$. Notice that an interval can be clamped as an integer if its lower bound equals its upper bound, i.e., $a = b$.

Given two integer intervals, $\tilde{x} = [a, b]$ and $\tilde{y} = [c, d]$, an adder based on this representation can deduce interval $\tilde{z} = \tilde{x} + \tilde{y} = [a + c, b + d]$. Let $\mathcal{X} = \langle x_1 \dots x_u \rangle$ and $\mathcal{Y} = \langle y_1 \dots y_v \rangle$ be two vectors of Boolean variables where $\tilde{x} \triangleq \mathcal{A}(X)$ and $\tilde{y} \triangleq \mathcal{A}(Y)$. Let $\mathcal{Z} = \langle z_1 \dots z_{u+v} \rangle$ perform their sum, where $\tilde{z} \triangleq \mathcal{A}(Z)$. This adder generates the following two types of clauses:

1. $(\neg x_i \vee \neg y_j \vee z_{i+j})$, where $0 \leq i \leq u$ and $0 \leq j \leq v$
2. $(x_{p+1} \vee y_{q+1} \vee \neg z_{p+q+1})$, where $0 \leq p \leq u$ and $0 \leq q \leq v$

The clause in type 1 indicates that $\tilde{z} \geq i + j$, and in contrast, the clause in type 2 indicates that $\tilde{z} \leq p + q$. With the same convention of the original totalizer encoding [11], we let special variables x_0, y_0 and z_0 be a constant value 1 (i.e., $x_0 = y_0 = z_0 = 1$), and let special variables x_{u+1}, y_{v+1} and z_{u+v+1} be another constant value 0 (i.e., $x_{u+1} = y_{v+1} = z_{u+v+1} = 0$). Notice that these variables are initialized to corresponding constant values, which means that they cannot be assigned during solving process, hence they are not included in the unary representation of integer intervals. Using these variables will never affect Definition 15, but it helps us to normalize the encoding of interval addition and to make generated clauses uniform.

Example 13. Given $\tilde{x} = [3, 5]$ in range 0..7 and $\tilde{y} = [2, 4]$ in range 0..6, with unary representation, we have respective assignments $\mathcal{X} = \langle 111**00 \rangle$ where $\tilde{x} \triangleq \mathcal{A}(X)$, and $\mathcal{Y} = \langle 11**00 \rangle$ where $\tilde{y} \triangleq \mathcal{A}(Y)$. Generated CNF formulae $\bigwedge_{i=0}^7 \bigwedge_{j=0}^6 (\neg x_i \vee \neg y_j \vee z_{i+j})$ and $\bigwedge_{p=0}^7 \bigwedge_{q=0}^6 (x_{p+1} \vee y_{q+1} \vee \neg z_{p+q+1})$ lead to assignment $\mathcal{Z} = \langle 11111***0000 \rangle$, which indicates $\tilde{z} = [5, 9]$ in range 0..13, where $\tilde{z} \triangleq \mathcal{A}(Z)$.

However, in this chapter, it is unnecessary to ensure that $\tilde{z} \leq p + q$ for the left terms of the PB constraints, because we are not concerned with encoding $\sum_{i=1}^m w_i b_i \geq k$. Hence we omit such clauses as those in $\tilde{z} \leq p + q$ (i.e., type 2) from our encoding. We only have the conjunction of all the clauses in type 1 that ensure that $\tilde{z} \geq i + j$ through their unary representations with convention $x_0 = y_0 = z_0 = 1$:

$$\phi = \bigwedge_{i=0}^u \bigwedge_{j=0}^v (\neg x_i \vee \neg y_j \vee z_{i+j})$$

Maintaining arc-consistency [46] on CSP is based on the constraint propagation that guarantees that every value in a domain is consistent with every constraint. The advantage of this representation is that it allows this addition to be encoded in such a way that UP, the fundamental mechanism used inside most SAT solvers, is able to make the expected inferences, even when some variables are unassigned. Therefore, its resulting CNF formula allows the UP to derive all of the consequences of every assignment with respect to *generalized arc consistency* (GAC) [31]. The original proof shows that TO encoding maintains GAC, which helps the SAT solver prune the search space sooner.

TOs, like converters, are in charge of transforming the input vectors into the corresponding output vectors that contain the same values but also satisfy the requirements of the unary representation of integer intervals, which are applied to

efficiently encode PB constraints. In essence, these TOs play the identical role as the *sorting and adder networks* [26, 72].

The basic TO encoding procedure is defined in Algorithm 6. The input is a vector of pairs of weight and auxiliary Boolean variable $\langle (w_1, b_1) \dots (w_m, b_m) \rangle$. If $m = 1$, TO generates a vector of Boolean variables $\mathcal{X} = \langle b_1 \dots b_1 \rangle$, which is constituted by w_1 bits of b_1 , i.e., $|\mathcal{X}| = w_1$, because the integer $(w_1 \times b_1)$ can be represented by corresponding assignment $\mathcal{A}(X)$. Then TO returns this $\mathcal{X}$ and a tautology. If $m > 1$, the parent TO divides input sequence $\langle (w_1, b_1) \dots (w_m, b_m) \rangle$ into $\langle (w_1, b_1) \dots (w_{\lfloor m/2 \rfloor}, b_{\lfloor m/2 \rfloor}) \rangle$ and $\langle (w_{\lfloor m/2 \rfloor + 1}, b_{\lfloor m/2 \rfloor + 1}) \dots (w_m, b_m) \rangle$, and recursively calls two children TOs that take as input either of the subsequences and the output of the respective vectors of Boolean variables $\mathcal{X}$ and $\mathcal{Y}$ and CNF formulae ϕ_x and ϕ_y . Then with an adder that performs the addition of $\mathcal{X}$ and $\mathcal{Y}$, vector of Boolean variables $\mathcal{Z} = \langle z_1 \dots z_{|\mathcal{X}|+|\mathcal{Y}|} \rangle$ and CNF formula $\phi_z = \bigwedge_{i=0}^{|\mathcal{X}|} \bigwedge_{j=0}^{|\mathcal{Y}|} (\neg x_i \vee \neg y_j \vee z_{i+j})$ are generated. For each recursion, TO returns a vector of Boolean variables $\mathcal{Z}$ and CNF formula $\phi_x \wedge \phi_y \wedge \phi_z$ until the feedback satisfies condition $m = 1$.

Algorithm 6: TO($\langle (w_1, b_1) \dots (w_m, b_m) \rangle$): return $(\mathcal{Z}_{sum}, \Phi)$

Input : A vector of pairs of weight and auxiliary variable, i.e., $\text{vec}\langle (w_i, b_i) \rangle$
Output : A vector of variables and a CNF formula, i.e., $(\mathcal{Z}_{sum}, \Phi)$

```

1 if  $m = 1$  then
2   new  $\text{vec}\langle \text{var} \rangle \mathcal{X} \leftarrow \langle b_1 \dots b_1 \rangle$  where  $(w_1 \times b_1) \triangleq \mathcal{A}(X)$  s.t.  $|\mathcal{X}| = w_1$ ;
3   return  $(\mathcal{X}, \top)$ ;
4 else
5    $(\mathcal{X}, \phi_x) \leftarrow \text{TO}(\langle (w_1, b_1) \dots (w_{\lfloor m/2 \rfloor}, b_{\lfloor m/2 \rfloor}) \rangle)$ ;
6    $(\mathcal{Y}, \phi_y) \leftarrow \text{TO}(\langle (w_{\lfloor m/2 \rfloor + 1}, b_{\lfloor m/2 \rfloor + 1}) \dots (w_m, b_m) \rangle)$ ;
7   new  $\text{vec}\langle \text{var} \rangle \mathcal{Z} \leftarrow \emptyset$ ,  $\text{vec}\langle \text{cla} \rangle \phi_z \leftarrow \top$ ;
8   for  $i \leftarrow 0$  to  $|\mathcal{X}|$  do
9     for  $j \leftarrow 0$  to  $|\mathcal{Y}|$  do
10       $\phi_z \leftarrow \phi_z \wedge (\neg x_i \vee \neg y_j \vee z_{i+j})$ ;
11      if  $i + j \neq 0$  and  $z_{i+j} \notin \mathcal{Z}$  then  $\mathcal{Z}.\text{PUSH}(z_{i+j})$ ;
12    end
13  end
14  return  $(\mathcal{Z}, \phi_x \wedge \phi_y \wedge \phi_z)$ ;
15 end

```

The TO encoding mechanism is illustrated in Figure 4.1. Each adder requires two input vectors of Boolean variables and outputs one. Thus, like a typical m players tournament that decides one winner and eliminates the rest among $m - 1$ games, m inputs need a total of $m - 1$ times of using an adder in the whole TO encoding procedure. Let s be the sum of all the input weights, i.e., $s = \sum_{i=1}^m w_i$. The final output includes the vector of Boolean variables $\mathcal{Z}_{sum} = \langle z_1 \dots z_s \rangle$, which ensures that assignment $\mathcal{A}(Z)$ exists s.t. $Z \subseteq \mathcal{Z}_{sum}$ that can represents the sum of interval $\tilde{z}$, and CNF formula $\Phi = \bigwedge_{i=1}^{m-1} \phi_i$, which ensures that the minimum in interval $\tilde{z}$, denoted as ζ equals the sum of the left terms of the PB constraint, i.e., $\zeta = \sum_{i=1}^m w_i b_i$.

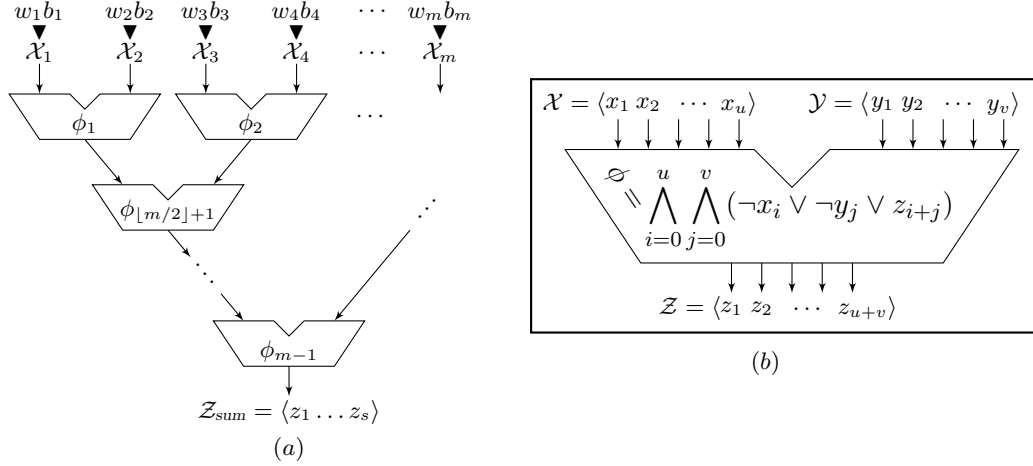


Figure 4.1. TO encoding: (a) Overview over binary tree of TO adders showing connection of input and output between each recursion. (b) Detailed adder for two input vectors of Boolean variables.

Proposition 1. *For any total truth assignment $\mathcal{A}$ of a weighted CNF (i.e., a WPMS instance), TO guarantees a final output that corresponds to an integer interval whose minimum equals the sum of the terms on the left of the PB constraint.*

To encode the entire PB constraint, we have to make a *comparator*, which generates CNF formula to guarantee that ζ is less than right term k , i.e., $\zeta < k$. In a straightforward way, the comparator generates this CNF by a *unit clause* $(\neg z_k)$, which constrains Boolean variable z_k to be assigned to 0. Notice that we always have $\exists z_k$ s.t. $z_k \in \mathcal{Z}_{sum}$, since $1 \leq k \leq s$. Formally, the PB constraint can be encoded into the CNF formula as follows:

$$\text{CNF}_{\text{PB}} \left(\sum_{i=1}^m w_i b_i < k \right) = \text{CNF}_{\text{TO}} \left(\sum_{i=1}^m w_i b_i = \zeta \right) \wedge \text{CNF}_{\text{comp}} (\zeta < k) = \Phi \wedge (\neg z_k)$$

Corollary 5. *Given an arbitrary PB constraint, the conjunction of the CNFs generated by TO and the comparator is satisfiable, if and only if this PB constraint is satisfiable.*

We can ignore and even remove Boolean variables x_i , y_i , and z_i such that $\forall i (i > k)$ from all generated inputs and outputs during TO's binary recursions, because they are never compared with k . The size of the CNF components (clauses, variables) encoded in TO can be reduced by omitting the parts related to these variables such that $\forall i (i > \kappa)$, where $\kappa = \sum_{\mathcal{A}(b_j)=1} w_j$, which is initialized by the total truth assignment $\mathcal{A}$ in the first run SAT solver on $\mathcal{F}_k$ without a PB constraint. This encoding technique is called κ -simplification (a.k.a. κ -cardinality networks) [8, 42], which guarantees that no redundancy of generated variables exists.

Example 14. Given a small PB constraint $3x + 2y < \kappa$ where $\kappa = 3$, we need to generate CNF formula $\Phi = \bigwedge_{i=0}^3 \bigwedge_{j=0}^2 (\neg x_i \vee \neg y_j \vee z_{i+j})$ and a unit clause (comparator) $(\neg z_3)$, in total 13 clauses. However, in Φ , some useless clauses which

are $(\neg x_i \vee \neg y_j \vee z_{i+j})$ where $i + j > \kappa$, are redundant. Therefore, with considering κ -simplification, we only need to generate CNF formula $\Phi = \bigwedge_{i=0}^3 \bigwedge_{j=0}^2 (\neg x_i \vee \neg y_j \vee z_{i+j})$ where $i + j \leq \kappa$, and comparator $(\neg z_3)$, in total 10 clauses.

Theorem 4. *For solving WPMS, TO always produces a polynomial-size CNF that requires $\mathcal{O}(m\kappa^2)$ clauses and $\mathcal{O}(m\kappa)$ auxiliary variables in m input Boolean variables with κ -simplification.*

Proof. Since the number of Boolean variables either of each input or output vector for each adder is bounded by κ , at most $(\kappa + 1)^2$ clauses are generated in ϕ . Hence, the total number of required clauses is bounded by $(m - 1) \times (\kappa + 1)^2$, which is in $\mathcal{O}(m\kappa^2)$, and the total number of generated auxiliary variables is bounded by $(m - 1) \times \kappa$, which is in $\mathcal{O}(m\kappa)$. $\square$

4.3.2 Weighted totalizer

Weighted totalizer (WTO) is a CNF encoding scheme that is heavily based on TO and variables partition by weighted combinations. As the same idea, the *generalized totalizer encoding* (GTE) [42] for PB constraints is also independently presented at almost the same period. Unlike other common methods, the number of auxiliary variables required for WTO is independent of the magnitudes of the weights, but it is related to how many unique sums of weights can be generated in each of its adders. If the unary representation of the integer intervals applied in TO is unweighted⁴, then WTO represents them using the *weighted unary representation*, because each input Boolean variable maps to its weight.

Mathematically, given a vector of pairs of weight and Boolean variable $\langle (w_1, x_1) \dots (w_m, x_m) \rangle$ whose containing weights are mutually different, we denote the set of the subscripts of each w that indicates its index in this vector by $I = \{1, \dots, m\}$. If we divide each pair into two sets W and X , where W stores all the unique weights and X stores the corresponding Boolean variables, then there exists two bijections: one is from X to W , denoted as $\omega(x_i) : x_i \mapsto w_i$, and another one is from W to I , denoted as $\iota(w_i) : w_i \mapsto i$, where $i \in \mathbb{N}, 1 \leq i \leq m$ for both bijections.

For the identical concern with TO, we also only need the conjunction of all the clauses of the type that ensure that each sum of intervals has its lower bound through their weighted unary representations with convention $x_0 = y_0 = z_0 = 1$, $\omega(x_0) = \omega(y_0) = \omega(z_0) = 0$, and $\iota(0) = 0$:

$$\phi = \bigwedge_{i=0}^{|\mathcal{X}|} \bigwedge_{j=0}^{|\mathcal{Y}|} (\neg x_i \vee \neg y_j \vee z_{\iota(\sigma)}), \text{ where } \omega(x_i) + \omega(y_j) = \omega(z_{\iota(\sigma)}) = \sigma \leq \kappa$$

Example 15. Given $\tilde{x} = \{i \in \mathbb{N} \mid i \geq 3\}$ and $\tilde{y} = \{i \in \mathbb{N} \mid i \geq 2\}$ both in range $0.. \kappa$ ($\kappa = 7$), with weighted unary representation, we have respective assignments $\mathcal{X} = \langle x_1 \rangle = \langle 1 \rangle$ where $\tilde{x} \triangleq \mathcal{A}(X)$, $\omega(x_1) = 3$ and $\iota(3) = 1$, and $\mathcal{Y} = \langle y_1 \rangle = \langle 1 \rangle$ where $\tilde{y} \triangleq \mathcal{A}(Y)$, $\omega(y_1) = 2$ and $\iota(2) = 1$. Generated CNF formula $\bigwedge_{i=0}^1 \bigwedge_{j=0}^1 (\neg x_i \vee \neg y_j \vee z_{\iota(\sigma)})$ where $\omega(x_i) + \omega(y_j) = \sigma \leq \kappa$, leads to assignment $\mathcal{Z} = \langle z_1 z_2 z_3 \rangle = \langle 111 \rangle$, which indicates $\tilde{z} = \{i \in \mathbb{N} \mid i \geq 5\}$ in range $0.. \kappa$, where $\tilde{z} \triangleq \mathcal{A}(Z)$, $\omega(z_1) = 2$, $\omega(z_2) =$

⁴“Unweighted” means that all the Boolean variables take a weight of 1.

3, $\omega(z_3) = 5$, $\iota(2) = 1$, $\iota(3) = 2$, and $\iota(5) = 3$. In contrast, as the way of TO, which is similar to Example 13 and 14, we have $\mathcal{X}' = \langle 1110000 \rangle$ and $\mathcal{Y}' = \langle 1100000 \rangle$. Generated CNF formula $\bigwedge_{i=0}^7 \bigwedge_{j=0}^7 (\neg x'_i \vee \neg y'_j \vee z'_{i+j})$ where $i + j < \kappa$, leads to assignment $\mathcal{Z}' = \langle 1111100 \rangle$, which indicates the same integer interval $\tilde{z}'$ as $\tilde{z}$, where $\tilde{z}' \triangleq \mathcal{A}(\mathcal{Z}')$.

The WTO procedure, which is defined in Algorithm 7, is heavily based on Algorithm 6. There are only two different places between them. The first place (see Line 2) explains that WTO only uses one Boolean variable to represent each input weight from the PB constraint, while TO uses the number of each weight of the Boolean variables. The second place (see Line 10 and 11) indicates that the above WTO adder, which means that WTO requires that number of generated Boolean variables for $\mathcal{Z}$ be less than or equal to the number of unique sums of weights, if with κ -simplification, while TO needs to generate a number of Boolean variables that is less than or equal to $|\mathcal{X}| + |\mathcal{Y}|$.

Algorithm 7: WTO($\langle (w_1, b_1) \dots (w_m, b_m) \rangle$): return $(\mathcal{Z}_{sum}, \Phi)$

Input : A vector of pairs of weight and auxiliary variable, i.e., $\text{vec}\langle (w_i, b_i) \rangle$
Output : A vector of variables and a CNF formula, i.e., $(\mathcal{Z}_{sum}, \Phi)$

```

1 if  $m = 1$  then
2   new  $\text{vec}\langle \text{var} \rangle \mathcal{X} \leftarrow \langle b_1 \rangle$ , where  $(w_1 \times b_1) \triangleq \mathcal{A}(X)$ ,  $\omega(b_1) = w_1$  and
    $\iota(w_1) = 1$  s.t.  $|\mathcal{X}| = 1$ ;
3   return  $(\mathcal{X}, \top)$ ;
4 else
5    $(\mathcal{X}, \phi_x) \leftarrow \text{WTO}(\langle (w_1, b_1) \dots (w_{\lfloor m/2 \rfloor}, b_{\lfloor m/2 \rfloor}) \rangle)$ ;
6    $(\mathcal{Y}, \phi_y) \leftarrow \text{WTO}(\langle (w_{\lfloor m/2 \rfloor + 1}, b_{\lfloor m/2 \rfloor + 1}) \dots (w_m, b_m) \rangle)$ ;
7   new  $\text{vec}\langle \text{var} \rangle \mathcal{Z} \leftarrow \emptyset$ ,  $\text{vec}\langle \text{cla} \rangle \phi_z \leftarrow \top$ ;
8   for  $i \leftarrow 0$  to  $|\mathcal{X}|$  do
9     for  $j \leftarrow 0$  to  $|\mathcal{Y}|$  do
10       $\phi_z \leftarrow \phi_z \wedge (\neg x_i \vee \neg y_j \vee z_{i+j})$ , where  $\omega(x_i) + \omega(y_j) = \sigma \leq \kappa$ ;
11      if  $\sigma \neq 0$  and  $z_{\iota(\sigma)} \notin \mathcal{Z}$  then  $\mathcal{Z}.\text{PUSH}(z_{\iota(\sigma)})$ ;
12    end
13  end
14  return  $(\mathcal{Z}, \phi_x \wedge \phi_y \wedge \phi_z)$ ;
15 end

```

With the same notation in TO, WTO finally outputs the vectors of Boolean variables $\mathcal{Z}_{sum}$ and CNF formula Φ . The comparator of WTO generates CNF formula $\Delta = \bigwedge_i \neg z_{\iota(i)}$ such that $\forall i (i \geq k)$, which constrains each Boolean variable $z_{\iota(i)}$ to be assigned to 0. Formally, the PB constraint can be encoded into CNF formula as follows:

$$\text{CNF}_{\text{PB}} \left(\sum_{i=1}^m w_i b_i < k \right) = \text{CNF}_{\text{WTO}} \left(\sum_{i=1}^m w_i b_i = \zeta \right) \wedge \text{CNF}_{\text{comp}} (\zeta < k) = \Phi \wedge \Delta$$

Since WTO behaves exactly as TO, the GAC property of TO holds for WTO as well. Notice that about the issue of space complexity, the best cases for WTO occur when

all of the weights are equal (i.e., the cases of cardinality constraints), in which the cases of the number of clauses and auxiliary variables are respectively $\mathcal{O}(m^2)$ and $\mathcal{O}(m \log_2(m))$ [11, 42]. For these best cases with κ -simplification, we have $\mathcal{O}(m\kappa)$ clauses and $\mathcal{O}(m \log_2(\kappa))$ variables [56]. However, in the worst cases where all of the weights are unequal, and all the sums (combinations) of the weight in both subsequences are different, the numbers of clauses and auxiliary variables required in WTO are exactly the same as those in TO respectively in $\mathcal{O}(m\kappa^2)$ and $\mathcal{O}(m\kappa)$.

4.4 N -level Modulo Totalizer Encoding

In this section, we propose another original encoding that is an extended WTO by applying modular arithmetic, called the n -level Modulo Totalizer (MTO), which can dramatically reduce the CNF size. To facilitate understanding, we present this scheme from a single-level modulo ($n = 1$) to a multi-level modulo ($n > 1$) in an orderly way.

4.4.1 Single-level modulo

We denote integer I by 1-level modulo number $q|_p r$ with modulus p such that $I = q \times p + r$, where $p \in \mathbb{N}$, $p > 1$. Here we name q , the quotient part, “upper digit”, and r , the remainder part, “lower digit”. Obviously, each weight owns its 1-level modulo number.

Lemma 1. *Let integer p be the modulus. Then the weighted unary representation satisfies the following properties:*

Split Any vector of Boolean variables $\mathcal{Z} = \langle z_1 \dots z_{|\mathcal{Z}|} \rangle$ where $\tilde{z} \triangleq \mathcal{A}(\mathcal{Z})$ can be split into two vectors $\mathcal{U} = \langle u_1 \dots u_{|\mathcal{U}|} \rangle$ and $\mathcal{L} = \langle l_1 \dots l_{|\mathcal{L}|} \rangle$, where $\tilde{u} \triangleq \mathcal{A}(\mathcal{U})$, $\tilde{l} \triangleq \mathcal{A}(\mathcal{L})$, and $\forall i (\omega(z_i) = \omega(u_j) \times p + \omega(l_k))$ s.t. $u_j \in \mathcal{U}$ and $l_k \in \mathcal{L}$.

Merger Any two vectors of Boolean variables $\mathcal{U} = \langle u_1 \dots u_{|\mathcal{U}|} \rangle$ and $\mathcal{L} = \langle l_1 \dots l_{|\mathcal{L}|} \rangle$, where $\tilde{u} \triangleq \mathcal{A}(\mathcal{U})$ and $\tilde{l} \triangleq \mathcal{A}(\mathcal{L})$ can be merged into one vector $\mathcal{Z} = \langle z_1 \dots z_{|\mathcal{Z}|} \rangle$, where $\tilde{z} \triangleq \mathcal{A}(\mathcal{Z})$ and $\forall i, j (\omega(u_i) \times p + \omega(l_j) = \omega(z_k))$ s.t. $z_k \in \mathcal{Z}$.

Integer intervals $\tilde{u}$ and $\tilde{l}$ respectively correspond to the upper and lower digits of the modulo number of the minimum in $\tilde{z}$. Hence we have $\tilde{u} \times p + \tilde{l} \triangleq \mathcal{A}(\mathcal{U}|_p \mathcal{L})$.

Example 16. Given $\tilde{z} = \{i \in \mathbb{N} \mid i \geq 5\}$, we have assignment $\mathcal{Z} = \langle z_1 z_2 z_3 \rangle = \langle *1 \rangle$, where $\tilde{z} \triangleq \mathcal{A}(\mathcal{Z})$, $\omega(z_1) = 2$, $\omega(z_2) = 3$, $\omega(z_3) = 5$, $\iota(2) = 1$, $\iota(3) = 2$ and $\iota(5) = 3$. Let modulus $p = 2$, because $2 = 1|_2 0$, $3 = 1|_2 1$ and $5 = 2|_2 1$, then $\mathcal{Z}$ can be split into $\mathcal{U}$ and $\mathcal{L}$. We have assignment $\mathcal{U} = \langle u_1 u_2 \rangle = \langle *1 \rangle$, which indicates $\tilde{u} = \{i \in \mathbb{N} \mid i \geq 2\}$, where $\tilde{u} \triangleq \mathcal{A}(\mathcal{U})$, $\omega(u_1) = 1$, $\omega(u_2) = 2$, $\iota(1) = 1$ and $\iota(2) = 2$, and assignment $\mathcal{L} = \langle l_1 \rangle = \langle 1 \rangle$, which indicates $\tilde{l} = \{i \in \mathbb{N} \mid i \geq 1\}$, where $\tilde{l} \triangleq \mathcal{A}(\mathcal{L})$, $\omega(l_1) = 1$ and $\iota(1) = 1$. So assignment $\mathcal{U}|_p \mathcal{L} = \langle u_1 u_2 \rangle|_p \langle l_1 \rangle = \langle *1 \rangle|_2 \langle 1 \rangle$ indicates $\{i \in \mathbb{N} \mid i \geq 5\}$, where $\tilde{u} \times p + \tilde{l} \triangleq \mathcal{A}(\mathcal{U}|_p \mathcal{L})$.

Theorem 5. *The minimum in integer interval $\tilde{z}$ is less than or equal to the minimum in integer interval $\tilde{u} \times p + \tilde{l}$. There also exists assignment $\mathcal{A}(\mathcal{U}|_p \mathcal{L})$ that can represent interval $\tilde{z}$.*

Proof. If $\min\{\tilde{z}\} = s$ ($s = q \times p + r$), then we always have $\min\{\tilde{u}\} = q$, $\min\{\tilde{l}\} = t \geq r$, and $\exists i$ s.t. $\omega(l_i) = r$. Thus we know that $\min\{\tilde{u} \times p + \tilde{l}\} = q \times p + t \geq s$. On the other hand, for the upper digits we have $\exists \mathcal{A}(U)$ s.t. $\mathcal{A}(u_{\iota(q)}) = 1$, and for the lower digits we have $\exists \mathcal{A}(L)$ s.t. $\mathcal{A}(l_{\iota(r)}) = 1$ and $\forall i \mathcal{A}(l_{\iota(i)}) = 0$ where $i > r$. As a consequence, we know that $\exists \mathcal{A}(U|_p L)$ s.t. $\tilde{z} \triangleq \mathcal{A}(U|_p L)$. $\square$

Given two integer intervals, $\tilde{x}$ and $\tilde{y}$, we have $\mathcal{U}_x|_p \mathcal{L}_x$ and $\mathcal{U}_y|_p \mathcal{L}_y$, which are the respective vectors of the Boolean variables that correspond to their upper and lower digits (e.g., $\mathcal{U}_x = \langle u_{x_i} \dots u_{x_{|\mathcal{U}_x|}} \rangle$ and $\mathcal{L}_x = \langle l_{x_i} \dots l_{x_{|\mathcal{L}_x|}} \rangle$) where $\tilde{x} \triangleq \mathcal{A}(U_x|_p L_x)$ and $\tilde{y} \triangleq \mathcal{A}(U_y|_p L_y)$. Let $\mathcal{U}_z|_p \mathcal{L}_z$ perform their sum, and then we construct an adder that can deduce interval $\tilde{u}_z \times p + \tilde{l}_z$ whose minimum is greater than or equal to the minimum in $\tilde{x} + \tilde{y}$, where $\tilde{u}_z \times p + \tilde{l}_z \triangleq \mathcal{A}(U_z|_p L_z)$. Due to the generation of the carry in modular arithmetic, the adder of a 1-level MTO is very different from that of WTO's. With convention $l_{x_0} = l_{y_0} = l_{z_0} = u_{x_0} = u_{y_0} = u_{z_0} = 1$, $\omega(l_{x_0}) = \omega(l_{y_0}) = \omega(l_{z_0}) = \omega(u_{x_0}) = \omega(u_{y_0}) = \omega(u_{z_0}) = 0$, $\iota(0) = 0$ and using Boolean variable c to indicate the carry, we have a conjunction of clauses for the lower digits:

$$\phi_l = \bigwedge_{i=0}^{|\mathcal{L}_x|} \bigwedge_{j=0}^{|\mathcal{L}_y|} \begin{cases} (\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{\iota(\sigma)}} \vee c), & \text{where } \omega(l_{x_i}) + \omega(l_{y_j}) = \sigma < p \\ (\neg l_{x_i} \vee \neg l_{y_j} \vee c), & \text{where } \omega(l_{x_i}) + \omega(l_{y_j}) = p \\ (\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{\iota(\sigma \bmod p)}}) \wedge (\neg l_{x_i} \vee \neg l_{y_j} \vee c), & \text{where } \omega(l_{x_i}) + \omega(l_{y_j}) = \sigma > p \end{cases}$$

and for the upper digits:

$$\phi_u = \bigwedge_{i=0}^{|\mathcal{U}_x|} \bigwedge_{j=0}^{|\mathcal{U}_y|} (\neg u_{x_i} \vee \neg u_{y_j} \vee u_{z_{\iota(\sigma)}}) \wedge (\neg c \vee \neg u_{x_i} \vee \neg u_{y_j} \vee u_{z_{\iota(\sigma+1)}}),$$

where $\omega(u_{x_i}) + \omega(u_{y_j}) = \sigma$

Example 17. Given $\tilde{x} = \tilde{y} = \{i \in \mathbb{N} \mid i \geq 2\}$ in range 0..10, let modulus $p = 3$, and we have two assignments, $\mathcal{U}_x|_p \mathcal{L}_x = \langle \rangle|_p \langle l_{x_1} l_{x_2} \rangle = \langle \rangle|_3 \langle *1 \rangle$ and $\mathcal{U}_y|_p \mathcal{L}_y = \langle \rangle|_p \langle l_{y_1} l_{y_2} \rangle = \langle \rangle|_3 \langle *1 \rangle$, where $\tilde{x} \triangleq \mathcal{A}(\mathcal{U}_x|_p L_x)$, $\tilde{y} \triangleq \mathcal{A}(\mathcal{U}_y|_p L_y)$, $\omega(l_{x_1}) = \omega(l_{y_1}) = 1$, $\omega(l_{x_2}) = \omega(l_{y_2}) = 2$, $\iota(1) = 1$ and $\iota(2) = 2$. Generated CNF formulae ϕ_l and ϕ_u lead to assignment $\mathcal{U}_z|_p \mathcal{L}_z = \langle u_{z_1} \rangle|_p \langle l_{z_1} l_{z_2} \rangle = \langle 1 \rangle|_3 \langle 1* \rangle$, which indicates $\tilde{z} = \{i \in \mathbb{N} \mid i \geq 4\}$ in range 0..10, where $\tilde{z} \triangleq \mathcal{A}(\mathcal{U}_z|_p L_z)$, $\omega(u_{z_1}) = \omega(l_{z_1}) = 1$, $\omega(l_{z_2}) = 2$, $\iota(1) = 1$ and $\iota(2) = 2$.

Notice that for lower digits, the adder generates clause $\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{\iota(\sigma)}} \vee c$, not $\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{\iota(\sigma)}}$, where $\omega(l_{x_i}) + \omega(l_{y_j}) = \sigma < p$. For example, consider only changing assignments $\mathcal{U}_x|_p \mathcal{L}_x$ and $\mathcal{U}_y|_p \mathcal{L}_y$ from $\langle \rangle|_3 \langle *1 \rangle$ to $\langle \rangle|_3 \langle 11 \rangle$ in example 7. If the adder generates clauses of type $\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{\iota(\sigma)}}$ instead of type $\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{\iota(\sigma)}} \vee c$, then l_{z_2} will be precisely assigned to 1, since $l_{x_1} \wedge l_{y_1} \rightarrow l_{z_2}$. As a result, there exists no assignment that can indicate that $\tilde{z} = \{i \in \mathbb{N} \mid i \geq 4\}$. Although clauses of type $\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{\iota(\sigma)}} \vee c$ lead to the minimum in $\tilde{z}$ that may become “more” greater than the minimum in the true sum of $\tilde{x} + \tilde{y}$, it guarantees the correct existence of $\mathcal{A}(\mathcal{U}_z|_p \mathcal{L}_z)$ s.t. $\tilde{x} + \tilde{y} \triangleq \mathcal{A}(U_z|_p L_z)$, which is not guaranteed by clauses of type $\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{\iota(\sigma)}}$.

Theorem 6. *Given two integer intervals, $\tilde{x}$ and $\tilde{y}$, we have $\mathcal{U}_x|_p\mathcal{L}_x$ and $\mathcal{U}_y|_p\mathcal{L}_y$, which are the respective vectors of the Boolean variables that correspond to their upper and lower digits, where $\tilde{x} \triangleq \mathcal{A}(U_x|_pL_x)$ and $\tilde{y} \triangleq \mathcal{A}(U_y|_pL_y)$. Let $\mathcal{U}_z|_p\mathcal{L}_z$ perform their sum, and then the 1-level MTO guarantees that there exists assignment $\mathcal{A}(U_z|_pL_z)$ s.t. $\tilde{x} + \tilde{y} \triangleq \mathcal{A}(U_z|_pL_z)$, and the minimum in interval $\tilde{u}_z \times p + \tilde{l}_z$ is greater than or equal to the minimum in interval $\tilde{x} + \tilde{y}$.*

Proof. If $\min\{\tilde{x}\} = a$ ($a = a_q \times p + a_r$) and $\min\{\tilde{y}\} = b$ ($b = b_q \times p + b_r$), then we always have $\min\{\tilde{u}_x\} = a_q$, $\min\{\tilde{l}_x\} = a_t \geq a_r$, $\exists i$ s.t. $\omega(l_{x_i}) = a_r$, $\min\{\tilde{u}_y\} = b_q$, $\min\{\tilde{l}_y\} = b_t \geq b_r$, and $\exists j$ s.t. $\omega(l_{y_j}) = b_r$. Therefore, we know that $\exists \mathcal{A}(U_z|_pL_z)$ s.t. $\mathcal{A}(u_{z_{i(a_q+b_q)}}) = 1$, $\mathcal{A}(l_{z_{i(a_r+b_r)}}) = 1$, $\forall i \mathcal{A}(u_{z_{i(i)}}) = 0$ and $\forall j \mathcal{A}(l_{z_{i(j)}}) = 0$ where $i > a_q + b_q$, $j > a_r + b_r$, which can represent interval $\tilde{x} + \tilde{y}$. Since $a \leq \min\{\tilde{u}_x \times p + \tilde{l}_x\} = a_q \times p + a_t$, $b \leq \min\{\tilde{u}_y \times p + \tilde{l}_y\} = b_q \times p + b_t$, and $\min\{\tilde{u}_z \times p + \tilde{l}_z\} \geq \min\{(\tilde{u}_x + \tilde{u}_y) \times p + (\tilde{l}_x + \tilde{l}_y)\} = (a_q + b_q) \times p + (a_t + b_t)$ caused by the generated clauses of type $\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{i(\sigma)}} \vee c$, we know that $\min\{\tilde{u}_z \times p + \tilde{l}_z\} \geq \min\{\tilde{x} + \tilde{y}\} = (a_q + b_q) \times p + (a_r + b_r)$. $\square$

The 1-level MTO encoding procedure is defined in Algorithm 8, and the final output includes the upper and lower vectors of Boolean variables $\mathcal{U}_z|_p\mathcal{L}_z$ and CNF formula Φ . The minimum in $\tilde{u}_z \times p + \tilde{l}_z$, denoted as ζ , is greater than or equal to the sum of the left terms of the PB constraint, i.e., $\zeta \geq \sum_{i=1}^m w_i b_i$.

Proposition 2. *For any total truth assignment $\mathcal{A}$ of a weighted CNF, 1-level MTO guarantees that the final output corresponds to an integer interval whose minimum is greater than or equal to the sum of the terms on the left of the PB constraint.*

We also have modulo number $k_u|_p k_l$ of the right term of PB constraint k , where $k = k_u \times p + k_l$. Since the upper digits have a higher priority for comparison than the lower digits, the comparator of the 1-level MTO generates CNF formula $\Delta = \bigwedge_i \neg u_{z_{i(i)}} \bigwedge_j (\neg u_{z_{i(k_u)}} \vee \neg l_{z_{i(j)}})$ such that $\forall i (i > k_u)$ and $\forall j (j \geq k_l)$. Formally, the PB constraint can be encoded into the CNF formula as follows:

$$\text{CNF}_{\text{PB}} \left(\sum_{i=1}^m w_i b_i < k \right) = \text{CNF}_{\text{1LvMTO}} \left(\sum_{i=1}^m w_i b_i \leq \zeta \right) \wedge \text{CNF}_{\text{comp}} (\zeta < k) = \Phi \wedge \Delta$$

Corollary 6. *Given an arbitrary PB constraint, the conjunction of the CNFs generated by the 1-level MTO and the comparator is satisfiable, if and only if this PB constraint is satisfiable.*

Theorem 7. *For solving WPMS, 1-level MTO always produces a polynomial-size CNF requiring $\mathcal{O}(4mp^2)$ clauses and $\mathcal{O}(2mp)$ auxiliary variables in m input Boolean variables with p -simplification, where modulus $p = \lceil \kappa^{\frac{1}{2}} \rceil$.*

Proof. Since the number of Boolean variables of either each input or output upper/lower vector for each adder is bounded by p , at most $2 \times (p+1)^2$ clauses are generated either in ϕ_l or in ϕ_u . Hence, the total number of required clauses is bounded by $(m-1) \times 4 \times (p+1)^2$, which is in $\mathcal{O}(4mp^2)$, and the total number of generated auxiliary variables is bounded by $(m-1) \times 2 \times p$, which is in $\mathcal{O}(2mp)$. $\square$

Algorithm 8: 1LvMTO($\langle\langle(w_1, b_1) \dots (w_m, b_m)\rangle, p\rangle$): return $(\mathcal{U}_z|_p \mathcal{L}_z, \Phi)$

Input : A vector of weight and variable pairs, and a modulus
 $\langle\langle(w_1, b_1) \dots (w_m, b_m)\rangle, p\rangle$

Output: Upper and lower vectors of Boolean variables and a CNF formula
 $(\mathcal{U}_z|_p \mathcal{L}_z, \Phi)$

```

1 if  $m = 1$  then
2   new vec $\langle\text{var}\rangle$   $\mathcal{U} \leftarrow \langle b_1 \rangle$  where  $\lfloor w_1/p \rfloor \times b_1 \triangleq \mathcal{A}(U)$ ,  $\omega(b_1) = \lfloor w_1/p \rfloor \neq 0$ 
   and  $\iota(\lfloor w_1/p \rfloor) = 1$  s.t.  $|\mathcal{U}| = 1$ ;
3   new vec $\langle\text{var}\rangle$   $\mathcal{L} \leftarrow \langle b_1 \rangle$  where  $(w_1 \bmod p) \times b_1 \triangleq \mathcal{A}(L)$ ,
    $\omega(b_1) = (w_1 \bmod p) \neq 0$  and  $\iota(w_1 \bmod p) = 1$  s.t.  $|\mathcal{L}| = 1$ ;
4   return  $(\mathcal{U}|_p \mathcal{L}, \top)$ ;
5 else
6    $(\mathcal{U}_x|_p \mathcal{L}_x, \phi_x) \leftarrow \text{1LvMTO}(\langle\langle(w_1, b_1) \dots (w_{\lfloor m/2 \rfloor}, b_{\lfloor m/2 \rfloor})\rangle, p)$ ;
7    $(\mathcal{U}_y|_p \mathcal{L}_y, \phi_y) \leftarrow \text{1LvMTO}(\langle\langle(w_{\lfloor m/2 \rfloor + 1}, b_{\lfloor m/2 \rfloor + 1}) \dots (w_m, b_m)\rangle, p)$ ;
8   new vec $\langle\text{var}\rangle$   $\mathcal{U}_z \leftarrow \emptyset$ ,  $\mathcal{L}_z \leftarrow \emptyset$ , vec $\langle\text{cla}\rangle$   $\phi_u \leftarrow \top$ ,  $\phi_l \leftarrow \top$ ;
    $\triangleright$  the adder part for the lower digits
9   for  $i \leftarrow 0$  to  $|\mathcal{L}_x|$  do
10    for  $j \leftarrow 0$  to  $|\mathcal{L}_y|$  do
11      if  $\omega(l_{x_i}) + \omega(l_{y_j}) < p$  then
12         $\phi_l \leftarrow \phi_l \wedge (\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{\iota(\sigma)}} \vee c)$ , where  $\omega(l_{x_i}) + \omega(l_{y_j}) = \sigma$ ;
13        if  $\sigma \neq 0$  and  $l_{z_{\iota(\sigma)}} \notin \mathcal{L}_z$  then  $\mathcal{L}_z.\text{PUSH}(l_{z_{\iota(\sigma)}})$ ;
14      else if  $\omega(l_{x_i}) + \omega(l_{y_j}) > p$  then
15         $\phi_l \leftarrow \phi_l \wedge (\neg l_{x_i} \vee \neg l_{y_j} \vee l_{z_{\iota(\sigma \bmod p)}}) \wedge (\neg l_{x_i} \vee \neg l_{y_j} \vee c)$ , where
         $\omega(l_{x_i}) + \omega(l_{y_j}) = \sigma$ ;
16        if  $\sigma \neq 0$  and  $l_{z_{\iota(\sigma \bmod p)}} \notin \mathcal{L}_z$  then  $\mathcal{L}_z.\text{PUSH}(l_{z_{\iota(\sigma \bmod p)}})$ ;
17      else  $\phi_l \leftarrow \phi_l \wedge (\neg l_{x_i} \vee \neg l_{y_j} \vee c)$ ;  $\triangleright \omega(l_{x_i}) + \omega(l_{y_j}) = p$ 
18    end
19  end
   $\triangleright$  the adder part for the upper digits
20  for  $i \leftarrow 0$  to  $|\mathcal{U}_x|$  do
21    for  $j \leftarrow 0$  to  $|\mathcal{U}_y|$  do
22       $\phi_u \leftarrow \phi_u \wedge (\neg u_{x_i} \vee \neg u_{y_j} \vee u_{z_{\iota(\sigma)}}) \wedge (\neg c \vee \neg u_{x_i} \vee \neg u_{y_j} \vee u_{z_{\iota(\sigma+1)}})$ ,
      where  $\omega(u_{x_i}) + \omega(u_{y_j}) = \sigma$ ;
23      if  $\sigma \neq 0$  and  $u_{z_{\iota(\sigma)}} \notin \mathcal{U}_z$  then  $\mathcal{U}_z.\text{PUSH}(u_{z_{\iota(\sigma)}})$ ;
24      if  $u_{z_{\iota(\sigma+1)}} \notin \mathcal{U}_z$  then  $\mathcal{U}_z.\text{PUSH}(u_{z_{\iota(\sigma+1)}})$ ;
25    end
26  end
27  new  $\phi_z \leftarrow \phi_u \wedge \phi_l$ ;
28  return  $(\mathcal{U}_z|_p \mathcal{L}_z, \phi_x \wedge \phi_y \wedge \phi_z)$ ;
29 end

```

4.4.2 Multi-level modulo

To generalize an MTO to a multi-level modulo, we denote integer I by n -level modulo number $d^{n+1}|_{\lambda_n} \dots d^3|_{\lambda_2} d^2|_{\lambda_1} d^1$ with the vector of moduli $\Lambda = \langle \lambda_1, \lambda_2, \dots, \lambda_n \rangle$ such that $I = d^1 + d^2 \times (\lambda_1) + d^3 \times (\lambda_1 \times \lambda_2) + \dots + d^{n+1} \times (\lambda_1 \times \lambda_2 \times \dots \times \lambda_n) = d^{n+1} \times \prod_{i=1}^n \lambda_i + \dots + d^3 \times \prod_{i=1}^2 \lambda_i + d^2 \times \prod_{i=1}^1 \lambda_i + d^1$, where $\lambda_i \in \mathbb{N}$, $\lambda_i > 1$. Note that the superscript of d indicates the order number of digits, e.g., d^i is the i -th digit. Here we name d^1 the “lowest digit” and d^h the “intermediate digit”, where $h \in \mathbb{N}$, $2 \leq h \leq n$, and we name d^{n+1} the “uppermost digit”. The notation of the n -level modulo number and the vector of moduli Λ are also respectively called *mixed radix representation* and *mixed radix base* [3]. Obviously, each weight has its own n -level modulo number.

Example 18. The 4-level modulo number of 1229223 is $2|_{70}|_{245}|_{60}27|_{60}3$, which equals 1,229,223 seconds, or 2 weeks, 5 hours, 27 minutes, and 3 seconds.

Lemma 2. Let $\Lambda = \langle \lambda_1 \dots \lambda_n \rangle$ be the vector of moduli. Then the weighted unary representation satisfies the following properties:

Split Any vector of Boolean variables $\mathcal{Z} = \langle z_1 \dots z_{|\mathcal{Z}|} \rangle$, where $\tilde{z} \triangleq \mathcal{A}(\mathcal{Z})$ can be split into $n+1$ vectors $\mathcal{D}^{n+1} = \langle d_1^{n+1} \dots d_{|\mathcal{D}^{n+1}|}^{n+1} \rangle, \dots, \mathcal{D}^1 = \langle d_1^1 \dots d_{|\mathcal{D}^1|}^1 \rangle$, where $\widetilde{d^{n+1}} \triangleq \mathcal{A}(D^{n+1}), \dots, \widetilde{d^1} \triangleq \mathcal{A}(D^1)$, and $\forall k (\omega(z_k) = \omega(d_{j_1}^{n+1}) \times \prod_{i=1}^n \lambda_i + \dots + \omega(d_{j_n}^2) \times \prod_{i=1}^1 \lambda_i + \omega(d_{j_{n+1}}^1))$ s.t. $d_{j_1}^{n+1} \in \mathcal{D}^{n+1}, \dots, d_{j_{n+1}}^1 \in \mathcal{D}^1$.

Merger Any $n+1$ vectors of Boolean variables $\mathcal{D}^{n+1} = \langle d_1^{n+1} \dots d_{|\mathcal{D}^{n+1}|}^{n+1} \rangle, \dots, \mathcal{D}^1 = \langle d_1^1 \dots d_{|\mathcal{D}^1|}^1 \rangle$, where $\widetilde{d^{n+1}} \triangleq \mathcal{A}(D^{n+1}), \dots, \widetilde{d^1} \triangleq \mathcal{A}(D^1)$ can be merged into one vector $\mathcal{Z} = \langle z_1 \dots z_{|\mathcal{Z}|} \rangle$ where $\tilde{z} \triangleq \mathcal{A}(\mathcal{Z})$ and $\forall j_1, \dots, j_{n+1} (\omega(d_{j_1}^{n+1}) \times \prod_{i=1}^n \lambda_i + \dots + \omega(d_{j_n}^2) \times \prod_{i=1}^1 \lambda_i + \omega(d_{j_{n+1}}^1) = \omega(z_j))$ s.t. $z_k \in \mathcal{Z}$.

Integer intervals $\widetilde{d^{n+1}}, \dots, \widetilde{d^1}$ respectively correspond to each digit of the n -level modulo number of the minimum in $\tilde{z}$. Hence we have $\widetilde{d^{n+1}} \times \prod_{i=1}^n \lambda_i + \dots + \widetilde{d^2} \times \prod_{i=1}^1 \lambda_i + \widetilde{d^1} \triangleq \mathcal{A}(D^{n+1}|_{\lambda_n} \dots |_{\lambda_1} D^1)$.

Theorem 8. The minimum in integer interval $\tilde{z}$ is less than or equal to the minimum in integer interval $\widetilde{d^{n+1}} \times \prod_{i=1}^n \lambda_i + \dots + \widetilde{d^2} \times \prod_{i=1}^1 \lambda_i + \widetilde{d^1}$. There also exists assignment $\mathcal{A}(D^{n+1}|_{\lambda_n} \dots |_{\lambda_1} D^1)$ that can represent interval $\tilde{z}$.

Given two integer intervals, $\tilde{x}$ and $\tilde{y}$, we have $\mathcal{D}_x^{n+1}|_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_x^1$ and $\mathcal{D}_y^{n+1}|_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_y^1$, which are the respective vectors of the Boolean variables corresponding to their digits (e.g., $\mathcal{D}_x^n = \langle d_{x_i}^n \dots d_{x_{|\mathcal{D}_x^n|}}^n \rangle$), where $\tilde{x} \triangleq \mathcal{A}(D_x^{n+1}|_{\lambda_n} \dots |_{\lambda_1} D_x^1)$ and $\tilde{y} \triangleq \mathcal{A}(D_y^{n+1}|_{\lambda_n} \dots |_{\lambda_1} D_y^1)$. Let $\mathcal{D}_z^{n+1}|_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_z^1$ perform their sum, and then we construct an adder that can deduce interval $\widetilde{d_z^{n+1}} \times \prod_{i=1}^n \lambda_i + \dots + \widetilde{d_z^2} \times \prod_{i=1}^1 \lambda_i + \widetilde{d_z^1}$ whose minimum is greater than or equal to the minimum in $\tilde{x} + \tilde{y}$, where $\widetilde{d_z^{n+1}} \times \prod_{i=1}^n \lambda_i + \dots + \widetilde{d_z^2} \times \prod_{i=1}^1 \lambda_i + \widetilde{d_z^1} \triangleq \mathcal{A}(D_z^{n+1}|_{\lambda_n} \dots |_{\lambda_1} D_z^1)$. With convention $d_{x_0}^{n+1} = d_{y_0}^{n+1} = d_{z_0}^{n+1} = \dots = d_{x_0}^1 = d_{y_0}^1 = d_{z_0}^1 = 1$, $\omega(d_{x_0}^{n+1}) = \omega(d_{y_0}^{n+1}) = \omega(d_{z_0}^{n+1}) = \dots = \omega(d_{x_0}^1) = \omega(d_{y_0}^1) = \omega(d_{z_0}^1) = 0$, $\iota(0) = 0$ and using Boolean variables $c^n \dots c^1$ to

respectively indicate the carries of $\mathcal{D}_z^n \dots \mathcal{D}_z^1$, we have the conjunction of the clauses for the lowest digits:

$$\phi^1 = \bigwedge_{i=0}^{|\mathcal{D}_x^1|} \bigwedge_{j=0}^{|\mathcal{D}_y^1|} \begin{cases} (\neg d_{x_i}^1 \vee \neg d_{y_j}^1 \vee d_{z_{\iota(\sigma)}}^1 \vee c^1), \text{ where } \omega(d_{x_i}^1) + \omega(d_{y_j}^1) = \sigma < \lambda_1 \\ (\neg d_{x_i}^1 \vee \neg d_{y_j}^1 \vee c^1), \text{ where } \omega(d_{x_i}^1) + \omega(d_{y_j}^1) = \lambda_1 \\ (\neg d_{x_i}^1 \vee \neg d_{y_j}^1 \vee d_{z_{\iota(\sigma \bmod \lambda_1)}}^1) \wedge (\neg d_{x_i}^1 \vee \neg d_{y_j}^1 \vee c^1), \\ \text{where } \omega(d_{x_i}^1) + \omega(d_{y_j}^1) = \sigma > \lambda_1 \end{cases} \quad (4.1)$$

and for the intermediate digits:

$$\phi^h = \bigwedge_{i=0}^{|\mathcal{D}_x^h|} \bigwedge_{j=0}^{|\mathcal{D}_y^h|} \begin{cases} (\neg d_{x_i}^h \vee \neg d_{y_j}^h \vee d_{z_{\iota(\sigma)}}^h \vee c^h), \text{ where } \omega(d_{x_i}^h) + \omega(d_{y_j}^h) = \sigma < \lambda_h \\ (\neg d_{x_i}^h \vee \neg d_{y_j}^h \vee c^h), \text{ where } \omega(d_{x_i}^h) + \omega(d_{y_j}^h) = \lambda_h \\ (\neg d_{x_i}^h \vee \neg d_{y_j}^h \vee d_{z_{\iota(\sigma \bmod \lambda_h)}}^h) \wedge (\neg d_{x_i}^h \vee \neg d_{y_j}^h \vee c^h), \\ \text{where } \omega(d_{x_i}^h) + \omega(d_{y_j}^h) = \sigma > \lambda_h \\ (\neg c^{h-1} \vee \neg d_{x_i}^h \vee \neg d_{y_j}^h \vee d_{z_{\iota(\sigma)}}^h \vee c^h), \\ \text{where } \omega(d_{x_i}^h) + \omega(d_{y_j}^h) + 1 = \sigma < \lambda_h \\ (\neg c^{h-1} \vee \neg d_{x_i}^h \vee \neg d_{y_j}^h \vee c^h), \text{ where } \omega(d_{x_i}^h) + \omega(d_{y_j}^h) + 1 = \lambda_h \\ (\neg c^{h-1} \vee \neg d_{x_i}^h \vee \neg d_{y_j}^h \vee d_{z_{\iota(\sigma \bmod \lambda_h)}}^h) \wedge (\neg c^{h-1} \vee \neg d_{x_i}^h \vee \neg d_{y_j}^h \vee c^h), \\ \text{where } \omega(d_{x_i}^h) + \omega(d_{y_j}^h) + 1 = \sigma > \lambda_h \end{cases} \quad (4.2)$$

and for the uppermost digits:

$$\phi^{n+1} = \bigwedge_{i=0}^{|\mathcal{D}_x^{n+1}|} \bigwedge_{j=0}^{|\mathcal{D}_y^{n+1}|} (\neg d_{x_i}^{n+1} \vee \neg d_{y_j}^{n+1} \vee d_{z_{\iota(\sigma)}}^{n+1}) \wedge (\neg c^n \vee \neg d_{x_i}^{n+1} \vee \neg d_{y_j}^{n+1} \vee d_{z_{\iota(\sigma+1)}}^{n+1}), \quad \text{where } \omega(d_{x_i}^{n+1}) + \omega(d_{y_j}^{n+1}) = \sigma \quad (4.3)$$

Theorem 9. *Given two integer intervals, $\tilde{x}$ and $\tilde{y}$, we have $\mathcal{D}_x^{n+1} |_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_x^1$ and $\mathcal{D}_y^{n+1} |_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_y^1$, which are the respective vectors of the Boolean variables that correspond to each digit where $\tilde{x} \triangleq \mathcal{A}(D_x^{n+1} |_{\lambda_n} \dots |_{\lambda_1} D_x^1)$ and $\tilde{y} \triangleq \mathcal{A}(D_y^{n+1} |_{\lambda_n} \dots |_{\lambda_1} D_y^1)$. Let $\mathcal{D}_z^{n+1} |_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_z^1$ perform their sum, and then the n -level MTO guarantees that there exists assignment $\mathcal{A}(\widetilde{\mathcal{D}_z^{n+1}} |_{\lambda_n} \dots |_{\lambda_1} \widetilde{\mathcal{D}_z^1})$ s.t. $\tilde{x} + \tilde{y} \triangleq \mathcal{A}(\widetilde{\mathcal{D}_z^{n+1}} |_{\lambda_n} \dots |_{\lambda_1} \widetilde{\mathcal{D}_z^1})$, and the minimum in interval $\widetilde{d_z^{n+1}} \times \prod_{i=1}^n \lambda_i + \dots + \widetilde{d_z^2} \times \prod_{i=1}^1 \lambda_i + \widetilde{d_z^1}$ is greater than or equal to the minimum in interval $\tilde{x} + \tilde{y}$.*

For avoiding cumbersome narratives, we omit proving Theorems 8 and 9 whose proofs are very similar to those of Theorems 5 and 6. The n -level MTO encoding procedure is defined in Algorithm 9, and the final output includes the lowest, intermediate, and uppermost vectors of Boolean variables $\mathcal{D}_z^{n+1} |_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_z^1$ and CNF formula Φ . The minimum in $\widetilde{d_z^{n+1}} \times \prod_{i=1}^n \lambda_i + \dots + \widetilde{d_z^2} \times \prod_{i=1}^1 \lambda_i + \widetilde{d_z^1}$, denoted as ζ , is greater than or equal to the sum of the terms on the left of the PB constraint, i.e., $\zeta \geq \sum_{i=1}^m w_i b_i$.

Algorithm 9: $n\text{LvMTO}(\langle (w_1, b_1) \dots (w_m, b_m) \rangle, \Lambda) : \text{return } (\mathcal{D}_z^{n+1}|_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_z^1, \Phi)$

Input : A vector of the weight and Boolean variable pairs, and a vector of moduli $(\langle (w_1, b_1) \dots (w_m, b_m) \rangle, \Lambda)$

Output : The uppermost, intermediate, and lowest vectors of Boolean variables and a CNF formula $(\mathcal{D}_z^{n+1}|_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_z^1, \Phi)$

```

1 if  $m = 1$  then
2   for  $i \leftarrow n + 1$  to 2 do  $\triangleright i$  is decremented in this loop
3     new vec $\langle \text{var} \rangle \mathcal{D}^i \leftarrow \langle b_1 \rangle$  where  $\lfloor w_1 / \prod_{j=1}^{i-1} \lambda_j \rfloor \times b_1 \triangleq \mathcal{A}(\mathcal{D}^i)$ ,
       $\omega(b_1) = \lfloor w_1 / \prod_{j=1}^{i-1} \lambda_j \rfloor \neq 0$  and  $\iota(\lfloor w_1 / \prod_{j=1}^{i-1} \lambda_j \rfloor) = 1$  s.t.  $|\mathcal{D}^i| = 1$ ;
4      $w_1 \leftarrow w_1 \bmod \prod_{j=1}^{i-1} \lambda_j$ ;
5   end
6   new vec $\langle \text{var} \rangle \mathcal{D}^1 \leftarrow \langle b_1 \rangle$  where  $w_1 \times b_1 \triangleq \mathcal{A}(\mathcal{D}^1)$ ,  $\omega(b_1) = w_1 \neq 0$  and
       $\iota(w_1) = 1$  s.t.  $|\mathcal{D}^1| = 1$ ;
7   return  $(\mathcal{D}_z^{n+1}|_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_z^1, \top)$ ;
8 else
9    $(\mathcal{D}_x^{n+1}|_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_x^1, \phi_x) \leftarrow n\text{LvMTO}(\langle (w_1, b_1) \dots (w_{\lfloor m/2 \rfloor}, b_{\lfloor m/2 \rfloor}) \rangle, \Lambda)$ ;
10   $(\mathcal{D}_y^{n+1}|_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_y^1, \phi_y) \leftarrow n\text{LvMTO}(\langle (w_{\lfloor m/2 \rfloor + 1}, b_{\lfloor m/2 \rfloor + 1}) \dots (w_m, b_m) \rangle, \Lambda)$ ;
11  new vec $\langle \text{var} \rangle \mathcal{D}_z^{n+1} \leftarrow \emptyset, \dots, \mathcal{D}_z^1 \leftarrow \emptyset$ ;
12  new vec $\langle \text{cla} \rangle \phi^{n+1} \leftarrow \top, \dots, \phi^1 \leftarrow \top$ ;
13   $\dots$ ;  $\triangleright$  omit the adder part for the lowest digits (Formula 4.1)
14  for  $h \leftarrow 2$  to  $n$  do
15     $\dots$ ;  $\triangleright$  omit the adder part for the intermediate digits (Formula 4.2)
16  end
17   $\dots$ ;  $\triangleright$  omit the adder part for the uppermost digits (Formula 4.3)
18  new  $\phi_z \leftarrow \phi^1 \wedge \dots \wedge \phi^{n+1}$ ;
19  return  $(\mathcal{D}_z^{n+1}|_{\lambda_n} \dots |_{\lambda_1} \mathcal{D}_z^1, \phi_x \wedge \phi_y \wedge \phi_z)$ ;
20 end

```

Proposition 3. *For any total truth assignment $\mathcal{A}$ of a weighted CNF, n -level MTO guarantees that final output corresponds to an integer interval whose minimum is greater than or equal to the sum of the terms on the left of the PB constraint.*

We also have n -level modulo number $k^{n+1}|_{\lambda_n} \dots |_{\lambda_1} k^1$ of the term on the right of PB constraint k , where $k = k^{n+1} \times \prod_{i=1}^n \lambda_i + \dots + k^2 \times \prod_{i=1}^1 \lambda_i + k^1$. By analyzing from the uppermost to the lowest digit, the n -level MTO comparator generates the following CNF formula:

$$\Delta = \bigwedge \begin{cases} \bigwedge_{i_1} \neg d_{z_{\iota(i_1)}}^{n+1} \\ \bigwedge_{i_2} (\neg d_{z_{\iota(k^{n+1})}}^{n+1} \vee \neg d_{z_{\iota(i_2)}}^n) \\ \vdots \\ \bigwedge_{i_n} (\neg d_{z_{\iota(k^{n+1})}}^{n+1} \vee \neg d_{z_{\iota(k^n)}}^n \vee \dots \vee \neg d_{z_{\iota(k^3)}}^3 \vee \neg d_{z_{\iota(i_n)}}^2) \\ \bigwedge_{i_{n+1}} (\neg d_{z_{\iota(k^{n+1})}}^{n+1} \vee \neg d_{z_{\iota(k^n)}}^n \vee \dots \vee \neg d_{z_{\iota(k^2)}}^2 \vee \neg d_{z_{\iota(i_{n+1})}}^1) \end{cases}$$

such that $\forall i_1 (i_1 > k^{n+1})$, $\forall i_2 (i_2 > k^n)$, $\dots$, $\forall i_n (i_n > k^2)$ and $\forall i_{n+1} (i_{n+1} \geq k^1)$.

Formally, the PB constraint can be encoded into the CNF formula as follows:

$$\text{CNF}_{\text{PB}} \left(\sum_{i=1}^m w_i b_i < k \right) = \text{CNF}_{n\text{LvMTO}} \left(\sum_{i=1}^m w_i b_i \leq \zeta \right) \wedge \text{CNF}_{\text{comp}} (\zeta < k) = \Phi \wedge \Delta$$

Corollary 7. *Given an arbitrary PB constraint, the conjunction of the CNFs generated by the n -level MTO and the comparator is satisfiable, if and only if this PB constraint is satisfiable.*

Theorem 10. *For solving WPMS, n -level MTO always produces a polynomial-size CNF requiring $\mathcal{O}(4nm\lambda_1^2)$ clauses and $\mathcal{O}(nm\lambda_1)$ auxiliary variables in m input Boolean variables with λ_1 -simplification, where the moduli satisfy that $\lambda_1 = \dots = \lambda_n = \lceil \kappa^{\frac{1}{n+1}} \rceil$.*

Proof. Since the number of the Boolean variables either of each input or the output vector of any digit for each adder is bounded by λ_1 , at most $2 \times (\lambda_1 + 1)^2$ clauses are generated either in ϕ^1 or in ϕ^{n+1} , and at most $4 \times (\lambda_1 + 1)^2$ clauses are generated in each ϕ^h , where $2 \leq h \leq n$. Hence, the total number of required clauses is bounded by $(m-1) \times (4 \times (\lambda_1 + 1)^2) + 4 \times (n-1) \times (\lambda_1 + 1)^2 = 4n(m-1)(\lambda_1 + 1)^2$, which is in $\mathcal{O}(4nm\lambda_1^2)$, and the total number of generated auxiliary variables is bounded by $(m-1) \times (n+1) \times \lambda_1$, which is in $\mathcal{O}(nm\lambda_1)$. $\square$

4.4.3 Selecting suitable mixed radix base

The mixed radix base has been widely applied to encode PB constraints, such as sorting-network encoding MINISAT⁺ [26]. Some techniques [1, 28] use *Residual Number System* and *Chinese Remainder Theorem* to derive a conjunction of the “PB modulo” constraints. These works minimized the sum of the digits of the weights when they are represented in a number system determined by an optimal mixed radix base [17]. In this section, we try to find a suitable, but probably not an optimal vector of moduli, i.e., a mixed radix base, for WPMS instances with low time-consumption because finding an optimal one becomes another $\mathcal{NP}$ -complete problem.

Consider the PB constraint given in Example 12. If we use a 4-level MTO to encode it, where $\lambda_1 = \dots = \lambda_4 = \lceil 2018^{\frac{1}{5}} \rceil = 5$, then we have $147 = 0|_5 1|_5 0|_5 4|_5 2$, $540 = 0|_5 4|_5 1|_5 3|_5 0$, $1365 = 2|_5 0|_5 4|_5 3|_5 0$ and $2018 = 3|_5 1|_5 0|_5 3|_5 3$. The above part of Table 4.1 shows the detailed calculation. For example, the third row shows the possible values of each digit of $w_2x_2 + w_3x_3$. The second digit (column d^2) of w_2 is 3 as well as that of w_3 . Ergo, the value of the second digit of $w_2x_2 + w_3x_3$ is 0 if $\mathcal{A}(x_2) = 0 \wedge \mathcal{A}(x_3) = 0$, 3 if $\mathcal{A}(x_2) = 1 \wedge \mathcal{A}(x_3) = 0$ or $\mathcal{A}(x_2) = 0 \wedge \mathcal{A}(x_3) = 1$, or 6 if $\mathcal{A}(x_2) = 1 \wedge \mathcal{A}(x_3) = 1$. Notice that the last value, 6, is greater than modulus $\lambda_2 = 5$ and produces value 1 for the current digit and carry c to the next digit, i.e., the third digit. Note that here we consider c a special value (0/1) that depends on the assignment of Boolean variable c^h generated by the 4-level MTO’s adder, where $c = 1$ if $\mathcal{A}(c^h) = 1$, and otherwise $c = 0$. Thus its possible values are 0, 1, 3 and c . We omit 0 in the table because it corresponds to the MTO feature in which setting all the output Boolean variables to 0 implies weight 0. If the values of the previous digit contains c , the carry must be taken into account for the current digit. For example, the value of

Table 4.1. Encoding PB constraint $147x_1 + 540x_2 + 1365x_3 < 2018$ by 4-level MTO with vectors of moduli $\Lambda_1 = \langle 5, 5, 5, 5 \rangle$ and $\Lambda_2 = \langle 3, 7, 5, 6 \rangle$

$$\Lambda_1 = \langle 5, 5, 5, 5 \rangle$$

	d^1	d^2	d^3	d^4	d^5
$w_1 = 147$	2	4	0	1	0
$w_2 = 540$	0	3	1	4	0
$w_3 = 1365$	0	3	4	0	2
$w_2x_2 + w_3x_3$	$\emptyset$	$1, 3, c$	$1, 2, 4, c$	$1, 4, c$	$2, 3$
$w_1x_1 + (w_2x_2 + w_3x_3)$	2	$1, 2, 3, 4, c$	$1, 2, 3, 4, c$	$1, 2, 3, 4, c$	$1, 2, 3, 4$
$\#Vars(d^h)$	2	11	11	10	7
$k = 2018$	3	3	0	1	3

$$\Lambda_2 = \langle 3, 7, 5, 6 \rangle$$

	d^1	d^2	d^3	d^4	d^5
$w_1 = 147$	0	0	2	1	0
$w_2 = 540$	0	5	0	5	0
$w_3 = 1365$	0	0	0	1	2
$w_2x_2 + w_3x_3$	$\emptyset$	5	$\emptyset$	$1, 5, c$	$2, 3$
$w_1x_1 + (w_2x_2 + w_3x_3)$	$\emptyset$	5	2	$1, 2, 5, c$	$2, 3, 4$
$\#Vars(d^h)$	0	3	2	10	6
$k = 2018$	2	0	1	1	3

its fourth digit becomes 0 if $\mathcal{A}(x_2) = 0 \wedge \mathcal{A}(c^4) = 0$, 1 if $\mathcal{A}(x_2) = 0 \wedge \mathcal{A}(c^4) = 1$, 4 if $\mathcal{A}(x_2) = 1 \wedge \mathcal{A}(c^4) = 0$, or 0 and produces c to the next digit if $\mathcal{A}(x_2) = 1 \wedge \mathcal{A}(c^4) = 1$. We can ignore the assignment of x_3 because the fourth digit of w_3 is 0. The second to last row shows the numbers of auxiliary variables required by the 4-level MTO. They correspond to the total numbers of digits in each column. For example, $\#Vars(d^4) = \#\{1\} + \#\{4\} + \#\{1, 4, c\} + \#\{1, 2, 3, 4, c\} = 1 + 1 + 3 + 5 = 10$.

It is important to find a suitable mixed radix base with low time-consumption that reduces the number of auxiliary variables for the n -level MTO. Reconsider the above example. Since the greatest common divisor of w_1 , w_2 , and w_3 is 3, in other words, the rate of divisibility by 3 is 100%, setting $\lambda_1 = 3$ leads to the amount of auxiliary variables required for the first digit to be 0. Before selecting the second modulus, we update all the weights to their quotients of dividing λ_1 , i.e., $w_1 = 49$, $w_2 = 180$, and $w_3 = 455$. This time, the rate of divisibility by 5 is 66.7% and by 7, because both w_2 and w_3 are divisible by 5, and both w_1 and w_3 are divisible by 7. For a faster convergence, the greater one, i.e., 7, is picked to be the second modulus. Then we repeatedly update all the weights, analyze each rate of divisibility by integer i such that $\forall i (2 \leq i \leq k)$, and select one corresponding to the highest rate of divisibility as the next modulus λ , until $\prod_{i=1}^4 \lambda_i \geq k$. We also give a detailed calculation of encoding the PB constraint with the vector of moduli $\Lambda_2 = \langle 3, 7, 5, 6 \rangle$ on the low part of Table 4.1. Clearly, the total number of auxiliary variables required by the 4-level MTO using Λ_2 is less than using $\Lambda_1 = \langle 5, 5, 5, 5 \rangle$. In fact, for selecting

a suitable vector of moduli, in addition to the idea mentioned above, we also add other heuristics tailored in our implementation, which are generally and mostly described in Algorithm 10, such as evaluating and voting for the candidates of λ (see Lines from 5 to 21), dynamically adjusting the lower limit of the required rate of divisibility (see Line 24), etc.

4.5 Implementation and Evaluation

We experimentally evaluated the TO, WTO, and n -level MTO performances in the newest version of QMAXSAT, which utilizes a SAT solver, GLUCOSE 3.0 [10, 9], as an inference engine, and a state-of-the-art WPMS solver, called MAXHS⁵ [21, 22, 23], by solving 767 instances used in the complete track of MaxSAT Evaluation 2017⁶. All experiments were performed on an Intel® Xeon (R) E3-1245 v6 3.70 GHz $\times$ 8 processor running Ubuntu 16.04 with a 3600-second timeout and a memory limit of 32 GB. Notice that MAXHS is also a satisfiability-based solver, but instead of encoding PB constraint into CNF, it moves the arithmetic optimization component of the WPMS problem off into a mixed integer programming (MIP) solver. The MIP solver is used to find optimal solutions (minimum cost hitting sets [21]) via *unsatisfiable core* which the SAT solver then test for feasibility. If the SAT solver is unable to find core any more, it means the WPMS solution has been found. Otherwise, the SAT solver returns a new core, the MIP solver re-optimizes, and the iterations continue. This hybrid approach is similar to Bender’s decomposition [38].

Table 4.2 shows the number (in brackets) of instances solved within their resource limitations and their average CPU times in seconds. For three different encodings, we also record that x instances suffered from memory-out and y instances suffered from time-out as “M: x , T: y ”. Each row shows the result of the corresponding instance group. The bottom row shows the total number of solved instances. We could not encode almost 60% of the 767 instances with TO or WTO because of the memory limit, but we only failed to encode two instances with n -level MTO, which means that it greatly reduced the amount of required memory. On the other hand, although MTO is slightly inferior to MAXHS in general, it benefited the most in the following groups: instance *haplotyping-pedigrees*, *shiftdesign*, *spot5*, *staff-scheduling*, *timetabling*, *causal-discovery*, *min-width*, and *css-refactoring*. N -level MTO does not hold GAC-preserving, since introducing carries destroys GAC. It is also one reason why TO or WTO shows better performance (faster) than the n -level MTO for some groups of instances, such as *af-synthesis*, *rna-alignment*, and *metro*.

Figure 4.2 shows the cumulative number (indicated by the horizontal axis) of the instances we encoded into the PB constraints within a given number (indicated by the vertical axis) of the variables or the clauses. For example, the graph on the right indicates that we respectively encoded about 200, 240, and 700 instances into the PB constraints with TO, WTO, and n -level MTO within 10^7 clauses. These two graphs support Table 4.2.

Figure 4.3 shows the cumulative number (indicated by the horizontal axis) of the instances solved within a given CPU time in seconds (indicated by the vertical

⁵ <http://www.maxhs.org>

⁶ <http://mse17.cs.helsinki.fi>

Algorithm 10: BASESELECTION($\langle (w_1, b_1) \dots (w_m, b_m) \rangle, \kappa$): return Λ

Input : A vector of the weight and variable pairs, and an integer $\kappa = \sum_{\mathcal{A}(b_i)=1} w_i$, w.r.t. κ -simplification $\langle (w_1, b_1) \dots (w_m, b_m) \rangle, \kappa$

Output : A vector of moduli Λ , where $\Lambda = \langle \lambda_1, \lambda_2, \dots \rangle$

```

1 int  $W, countZero, maxTime, Product \leftarrow 1$ ;
2 int  $vote[\kappa + 1], candidates[\kappa + 1]$ ;
3 new vec<int>  $\Lambda \leftarrow \emptyset, \Xi \leftarrow \emptyset$ ;
4 repeat
    ▷ heuristics 1: evaluating and voting for the candidates
5   forall the  $i$  ( $0 \leq i \leq \kappa$ ) do  $candidates[i] \leftarrow 0$ ;
6    $countZero \leftarrow 0, W \leftarrow 1$ ;
7   for  $i \leftarrow 1$  to  $m$  do
8     forall the  $j$  ( $0 \leq j \leq \kappa$ ) do  $vote[j] \leftarrow 0$ ;
9     if  $w_i \neq 0$  then  $W \leftarrow w_i$ ;
10    else  $countZero++$ ;
11    while  $W \neq 1$  do
12      for  $j \leftarrow \kappa$  to 2 do ▷  $j$  is decremented in this loop
13        if  $W \bmod j = 0$  then
14           $W = \lfloor W/j \rfloor$ ;  $vote[j]++$ ; break;
15        end
16      end
17    end
18    for  $j \leftarrow 2$  to  $\kappa$  do
19      if  $vote[j] > 0$  and  $|\Lambda| + 1 \geq \log_j(\kappa) \div j$  then  $candidates[j]++$ ;
20    end
21  end
22   $maxTime \leftarrow \max_i(candidates[i]); \Xi \leftarrow \arg \max_i(candidates[i])$ ;
23  foreach  $i$  ( $i \in \Xi$ ) in descending order do
    ▷ heuristics 2: dynamically adjusting the lower limit of the required
    rate of divisibility, i.e.,  $maxTime \div (m - countZero)$ 
24    if  $\kappa \div Product \geq i$  and  $\log_i(\frac{\kappa}{Product}) > \frac{m - countZero}{maxTime}$  then
25       $\lambda \leftarrow i$ ; break;
26    end
27  end
28  if  $\lambda \leq 1$  then break;
29  else  $\Lambda.PUSH(\lambda)$ ;
30  for  $i \leftarrow 1$  to  $m$  do  $w_i \leftarrow \lfloor w_i / \lambda \rfloor$ ;
31   $Product = \prod_i \lambda_i$  where  $\lambda_i \in \Lambda$ ;
32 until  $Product \geq \kappa$ ;
33 return  $\Lambda$ ;

```

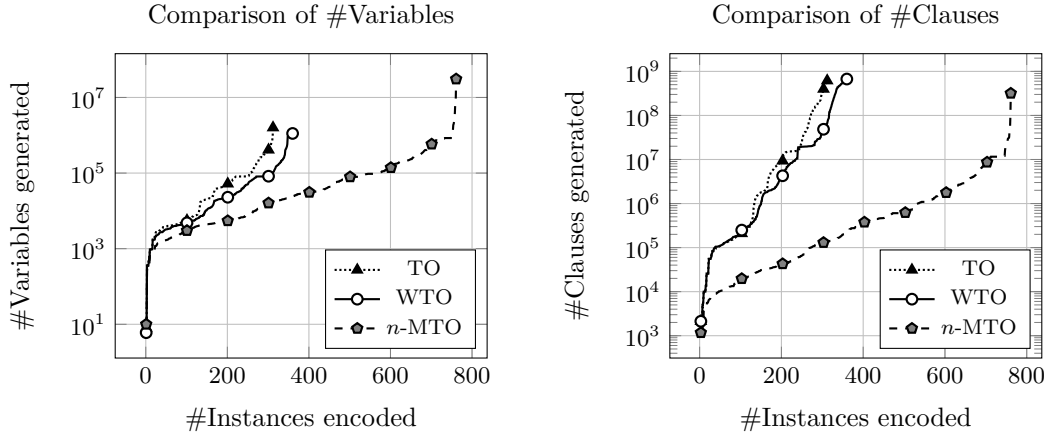
Table 4.2. Average runtimes and number of instances solved

Group	#Ins.	TO	WTO	<i>n</i> -level MTO	MaxHS
<i>auc-paths</i>	20	0.0 (0) M: 20, T: 0	0.0 (0) M: 20, T: 0	482.37 (16) M: 0, T: 4	0.04 (20)
<i>auc-scheduling</i>	20	0.0 (0) M: 20, T: 0	390.52 (3) M: 17, T: 0	0.91 (20)	0.04 (20)
<i>af-synthesis</i>	35	942.96 (34) M: 0, T: 1	1250.74 (31) M: 0, T: 4	1155.49 (34) M: 0, T: 1	1663.67 (7)
<i>biorepair</i>	30	13.24 (30)	13.34 (30)	3.21 (30)	1.99 (30)
<i>haplotyping-pedigrees</i>	35	0.0 (0) M: 35, T: 0	45.63 (16) M: 19, T: 0	46.48 (35)	97.71 (32)
<i>rna-alignment</i>	35	23.16 (35)	21.62 (35)	40.15 (35)	273.60 (21)
<i>maxcut</i>	30	615.67 (4) M: 5, T: 21	520.76 (3) M: 5, T: 22	479.31 (6) M: 0, T: 24	367.86 (19)
<i>csg</i>	10	0.0 (0) M: 10, T: 0	18.34 (2) M: 8, T: 0	19.99 (10)	6.08 (10)
<i>hs-timetabling</i>	14	1048.29 (1) M: 4, T: 9	827.18 (1) M: 4, T: 9	467.71 (1) M: 0, T: 13	870.45 (2)
<i>lisbon-wedding</i>	24	0.0 (0) M: 0, T: 24	0.0 (0) M: 0, T: 24	0.0 (0) M: 0, T: 24	0.0 (0)
<i>metro</i>	30	32.46 (30)	31.55 (30)	39.69 (30)	165.29 (30)
<i>miplib</i>	12	363.46 (4) M: 5, T: 3	41.41 (4) M: 4, T: 4	9.94 (5) M: 0, T: 7	67.07 (5)
<i>planning</i>	30	51.50 (26) M: 4, T: 0	4.48 (29) M: 1, T: 0	0.40 (30)	0.27 (30)
<i>railway-transport</i>	11	521.36 (5) M: 1, T: 5	418.01 (5) M: 1, T: 5	385.87 (5) M: 0, T: 6	75.92 (3)
<i>shiftdesign</i>	22	0.0 (0) M: 22, T: 0	0.0 (0) M: 22, T: 0	78.96 (22)	933.26 (14)
<i>spot5</i>	35	25.68 (12) M: 23, T: 0	90.72 (28) M: 6, T: 1	68.46 (35)	381.45 (22)
<i>staff-scheduling</i>	12	4.68 (1) M: 5, T: 6	1129.88 (3) M: 5, T: 4	1172.92 (4) M: 0, T: 8	0.96 (1)
<i>timetabling</i>	35	824.80 (13) M: 11, T: 11	571.13 (13) M: 11, T: 11	291.64 (24) M: 0, T: 11	376.60 (22)
<i>warehouses</i>	18	0.19 (1) M: 17, T: 0	0.08 (1) M: 17, T: 0	1178.85 (12) M: 0, T: 6	0.26 (18)
<i>BTBNSL</i>	35	0.0 (0) M: 35, T: 0	0.0 (0) M: 35, T: 0	1093.83 (20) M: 0, T: 15	229.27 (25)
<i>causal-discovery</i>	35	0.0 (0) M: 35, T: 0	0.0 (0) M: 35, T: 0	524.94 (29) M: 0, T: 6	315.50 (24)
<i>correlation-clustering</i>	35	0.0 (0) M: 35, T: 0	0.0 (0) M: 35, T: 0	1060.14 (3) M: 0, T: 32	349.50 (24)
<i>min-width</i>	35	0.0 (0) M: 35, T: 0	0.0 (0) M: 35, T: 0	325.08 (8) M: 0, T: 27	0.12 (3)
<i>relational-inference</i>	8	0.0 (0) M: 8, T: 0	0.0 (0) M: 8, T: 0	0.0 (0) M: 2, T: 6	512.46 (8)

Table 4.2 continued on next page

Table 4.2 continued from previous page

Group	#Ins.	TO	WTO	n -level MTO	MaxHS
<i>abstraction-refinement</i>	11	0.0 (0) M: 11, T: 0	1529.42 (3) M: 7, T: 1	863.71 (4) M: 0, T: 7	679.84 (11)
<i>frb</i>	34	19.41 (34)	19.37 (34)	18.01 (34)	16.00 (34)
<i>css-refactoring</i>	11	10.55 (1) M: 10, T: 0	9.70 (1) M: 10, T: 0	208.23 (9) M: 0, T: 2	108.46 (6)
<i>daculus</i>	35	0.0 (0) M: 35, T: 0	0.0 (0) M: 35, T: 0	5.27 (22) M: 0, T: 13	9.98 (35)
<i>mancoosi</i>	35	0.0 (0) M: 35, T: 0	0.0 (0) M: 35, T: 0	663.90 (21) M: 0, T: 14	0.96 (35)
<i>packup</i>	35	0.0 (0) M: 35, T: 0	0.0 (0) M: 35, T: 0	478.34 (30) M: 0, T: 5	0.80 (35)
Total	767	237.48 (231) M: 456, T: 80	243.39 (272) M: 410, T: 85	307.26 (534) M: 2, T: 231	182.28 (546)

**Figure 4.2.** Comparison of cumulative #Instances solved in generated #variables and #clauses

axis) with three encodings and MAXHS. For example, the graph shows that we respectively solved within 600 seconds about 200, 230, 450, and 500 instances with TO, WTO, n -level MTO, and MAXHS. Clearly, n -level MTO is superior to the other two encodings, followed closely by MAXHS.

Figure 4.4 compares the n -level MTO and MAXHS. Each point in the scatterplots represents a problem instance. Let its coordinate be (x, y) , where x indicates the CPU time in seconds for solving the instance with a n -level MTO, and y indicates the time with MAXHS. Thus, the points above the diagonal line indicate that the n -level MTO outperforms MAXHS for solving the corresponding instances. Interestingly, Figure 4.4 shows that they have strong complementarity. For example, the scatter points, which gather in the front of the top side bound of the value 3600 CPU seconds, represent that these instances cannot be solved by MAXHS, but take mapping in relatively short runtimes to be solved by n -level MTO.

In summary, the performance of QMAXSAT solver can be improved mostly by

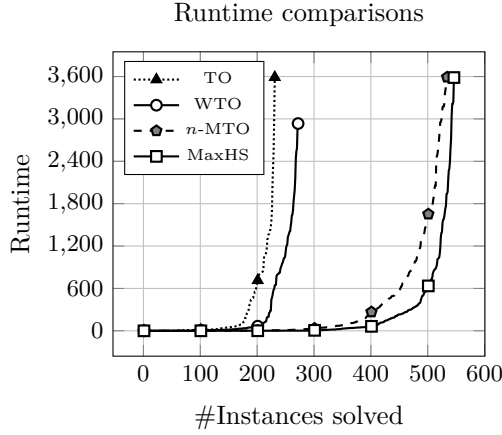


Figure 4.3. Comparison of cumulative #Instances solved within runtime

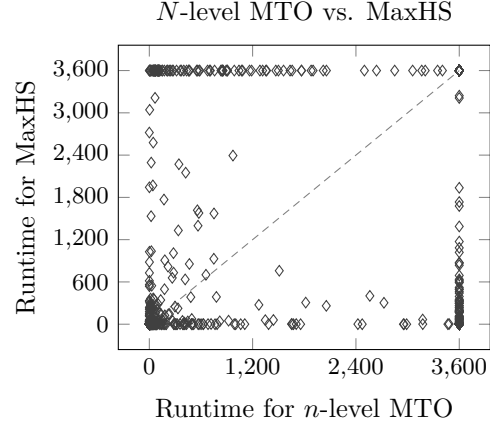


Figure 4.4. Comparison of runtimes of each instance solved with n -level MTO and MaxHS

more and more efficient CNF encodings of PB constraints. However, MaxHS displays superior performance on more benchmarks, probably because it is accomplished by exploiting extracted unsatisfiable cores and moving all of the arithmetic reasoning into a separate hitting set problem which can then be solved with techniques better suited to numeric reasoning, e.g. techniques from mathematical programming.

4.6 Related Works

The open question was answered [13, 52] about the existence of a polynomial-size encoding from PB constraints into CNF that maintains GAC through UP. Bailleux et al. proposed watchdog encoding [13], which has two different versions: one that detects inconsistencies and another that maintains GAC. The latter is the first SAT encoding of a PB constraint of m variables with maximum weight w_{max} , which is both polynomial ($\mathcal{O}(m^3 \log_2(m) \log_2(w_{max}))$ clauses and $\mathcal{O}(m^2 \log_2(m) \log_2(w_{max}))$ auxiliary variables). Recently, watchdog encoding has been generalized to a more compact framework with binary merger encoding by Manthey et al. [52]. Using a different translation of the components of watchdog encoding, binary merger encoding not only allows UP to maintain GAC but it can be further reduced to a space complexity of $\mathcal{O}(m^2 \log_2^2(m) \log_2(w_{max}))$ clauses. If GAC is not preserved, Binary Merger encoding requires only $\mathcal{O}(m \log_2^2(m) \log_2(w_{max}))$ clauses.

More interestingly, Abío et al. [2] overcame a possible exponential blowup of binary decision diagrams (BDD) by coefficient decomposition and proposed the first polynomial ($\mathcal{O}(m^3 \log_2(w_{max}))$ clauses and auxiliary variables) GAC-preserving reduced ordered binary decision diagram (ROBDD) based encoding for PB constraints. The other BDD-based approaches [12, 26] share great similarity with GTE, such as GAC and independence from the magnitude of the weight, which can be exponential in the worst case scenarios.

The idea of detecting inconsistency and maintaining GAC by UP was also

considered in sequential weight counter (SWC) encoding [37], proposed by Hölldobler et al., which generalizes sequential counter encoding [66] proposed by Sinz. For a PB constraint of the form, $\sum_{i=1}^m w_i x_i \leq k$, SWC encoding requires $\mathcal{O}(mk)$ clauses and auxiliary variables, which are linear in size w.r.t. the number of input variables. Nevertheless, like BDD-based approaches and GTE, SWC can also be exponential in the worst case.

Chapter 5

Practical Problems

Coalitional games are abstract models of cooperation that have been scrutinized in AI and multi-agent systems (MAS) research communities. A coalition of agents can sometimes accomplish things that individual agents cannot do or perhaps do them more efficiently. The coalition structure generation (CSG) problem is one major research topic in coalitional games, which involves partitioning a set of agents into coalitions to maximize social surplus, that is, the sum of the rewards of all coalitions [16]. There are various real CSG applications, such as distributed vehicle routing [61], multi-sensor networks [20], and so on. The CSG problem is equivalent to a complete set partition problem [74], and many MAS researchers have developed efficient algorithms to solve it.

There are two major types of the coalitional games: characteristic function games (CFGs) and partition function games (PFGs). CFGs are coalitional games with no externalities in which the value of a coalition is determined based on its members. On the other hand, PFGs are coalitional games with externalities in which the value of a coalition in partition function games depends on how the other agents are partitioned. We study CSG problems in PFGs. While various CSG algorithms for CFGs have been developed by many MAS researchers, there are few CSG algorithms for PFGs. However, in practice, CSG problems for coalitional games with externalities have been receiving greater attention. For example, consider restructuring and consolidating multiple companies in a market. If several competitors are merged into a single entity, the competitors of this new large company might lose sales. The market must consider the best way of restructuring the merged companies to maximize expected sales. Such a problem is represented as a PFG. In PFGs, the input of a coalitional game is a black-box function called a partition function that maps an embedded coalition (a coalition and a coalition structure containing the coalition) to its value. Representing an arbitrary partition function explicitly requires $\Theta(n^n)$ numbers, which are prohibitive for a large n . Recently, several concise representation schemes for a partition function have been proposed [53, 58, 67].

In this chapter, we utilize a graphical representation of PFGs called partition decision trees (PDTs) proposed by Skibski et al. [67]. In PDTs, multiple rules are graphically represented using rooted directed trees. Each rule consists of a condition for the partitioning agents and their positive/negative values. We propose the following new methods to solve CSG problems by PDTs representations: (1) a

depth-first branch-and-bound (DFBnB) algorithm for PDTs, (2) a DFBnB algorithm for modified PDTs, and (3) MaxSAT encoding. We experimentally show that our algorithm can reasonably solve CSG problems.

5.1 Related Works

The model of coalitional games with externalities (i.e., PFGs) [69], faces many computational challenges. PFGs require prohibitive time and space to represent the game. Since a coalition's value depends on the coalition structure to which it belongs, thus we must take into account all possible coalition structures. So the complexity of representing PFGs is $\Omega(n^{\frac{n}{2}})$ and $O(n^n)$, where n is the number of agents [60].

For concise representation, Michalak et al. [53] proposed embedded marginal contribution nets (embedded MC-nets) by extending marginal contribution nets (MC-nets) [39], which represent characteristic function games (CFGs) to handle externalities. Michalak et al. [54] developed a concise representation called weighted MC-nets, which was also an extension of MC-nets. Recently, Skibski et al. [67] proposed another representation—PDTs, and developed efficient algorithms that compute the extensions of the Shapley value in polynomial time. In Section 5.3, we precisely explain PDTs.

Rahwan et al. [58] first considered the CSG problem in PFGs, but their problem settings were the restricted classes of PFGs, where only positive or negative externalities exist. Ueda et al. [71] developed mixed integer programming (MIP) formulations for CSG problems in PFGs, which were represented by embedded MC-nets. In their settings, the number of agents equaled the number of rules, which was at most 150. Thus, their experimental setting was sparse in terms of the number of embedded MC-nets rules to evaluate against the number of all possible rules.

Liao et al. presented two MaxSAT encodings for CSGs represented by MC-nets or embedded MC-nets: agent-based and rule-based [48, 50, 49]. The former encodes agent relations into propositional logic while the latter encodes a rule relation-based approach [57, 71]. Their experimental results show that agent-based encoding is more time efficient than a relation-based approach. In this chapter, we present MaxSAT encoding with the agent-based approach.

5.2 Preliminaries

Let $N = \{1, 2, \dots, n\}$ be a set of agents. Coalition structure CS is defined as a partition of N into disjoint and exhaustive coalitions.

Definition 16 (Coalition structure). Coalition structure $CS = \{S_1, S_2, \dots\}$ satisfies the following conditions:

$$\forall i, j (i \neq j), S_i \cap S_j = \emptyset, \bigcup_{S_i \in CS} S_i = N.$$

Denote the set of all coalition structures as $\Pi(N)$.

In this chapter, we study coalitional games with externalities, called PFGs, in which the value of a coalition depends on how other agents have been partitioned.

Definition 17 (Embedded coalition). An *embedded coalition* consists of a pair of (S, CS) , where $S \in CS \in \Pi(N)$. The set of all embedded coalitions over N is denoted by $M(N)$, i.e., $M := \{(S, CS) : CS \in \Pi(N), S \in CS\}$.

Next we define a partition function.

Definition 18 (Partition function). A *partition function* assigns a real-valued payoff to every embedded coalition: $w : M \rightarrow \mathbb{R}$, where $w(\emptyset, CS) = 0$ for all CS .

For any coalition structure $CS \in \Pi(N)$, the value of CS is calculated by

$$W(CS) = \sum_{S_i \in CS} w(S_i, CS).$$

In general, a grand coalition that contains every agent in a single coalition is not always an optimal coalition structure. Thus, we have to find the optimal coalition structure that maximizes the sum of the values of partition functions. We define a CSG problem in partition function games as follows.

Definition 19 (CSG problem). For set of agents N , a CSG problem is to find optimal coalition structure CS^* such that

$$CS^* \in \arg \max_{CS \in \Pi(N)} W(CS).$$

Here we show an example of PFGs.

Example 19. Assume four agents, $N = \{a_1, a_2, a_3, a_4\}$. A partition function is given as follows:

$$\begin{aligned} w(\{a_1\}, \{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\}) &= 1, & w(\{a_2\}, \{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\}) &= 1, \\ w(\{a_3\}, \{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\}) &= 0, & w(\{a_4\}, \{\{a_1\}, \{a_2\}, \{a_3\}, \{a_4\}\}) &= 1, \\ w(\{a_1, a_2\}, \{\{a_1, a_2\}, \{a_3\}, \{a_4\}\}) &= 0, & w(\{a_3\}, \{\{a_1, a_2\}, \{a_3\}, \{a_4\}\}) &= 0, \\ w(\{a_4\}, \{\{a_1, a_2\}, \{a_3\}, \{a_4\}\}) &= 0, & w(\{a_1\}, \{\{a_1\}, \{a_2, a_4\}, \{a_3\}\}) &= 0, \\ w(\{a_2, a_4\}, \{\{a_1\}, \{a_2, a_4\}, \{a_3\}\}) &= 0, & w(\{a_3\}, \{\{a_1\}, \{a_2, a_4\}, \{a_3\}\}) &= 0, \\ w(\{a_1\}, \{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}) &= 1, & w(\{a_2\}, \{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}) &= 1, \\ w(\{a_3, a_4\}, \{\{a_1\}, \{a_2\}, \{a_3, a_4\}\}) &= 3, & w(\{a_1, a_3\}, \{\{a_1, a_3\}, \{a_2\}, \{a_4\}\}) &= 1, \\ w(\{a_2\}, \{\{a_1, a_3\}, \{a_2\}, \{a_4\}\}) &= 1, & w(\{a_4\}, \{\{a_1, a_3\}, \{a_2\}, \{a_4\}\}) &= 1, \\ w(\{a_1\}, \{\{a_1\}, \{a_2, a_3\}, \{a_4\}\}) &= 1, & w(\{a_2, a_3\}, \{\{a_1\}, \{a_2, a_3\}, \{a_4\}\}) &= 1, \\ w(\{a_4\}, \{\{a_1\}, \{a_2, a_3\}, \{a_4\}\}) &= 1, & w(\{a_1, a_4\}, \{\{a_1, a_4\}, \{a_2\}, \{a_3\}\}) &= 2, \\ w(\{a_2\}, \{\{a_1, a_4\}, \{a_2\}, \{a_3\}\}) &= -1, & w(\{a_3\}, \{\{a_1, a_4\}, \{a_2\}, \{a_3\}\}) &= 3, \\ w(\{a_1\}, \{\{a_1\}, \{a_2, a_3, a_4\}\}) &= 0, & w(\{a_2, a_3, a_4\}, \{\{a_1\}, \{a_2, a_3, a_4\}\}) &= 2, \\ w(\{a_2\}, \{\{a_2\}, \{a_1, a_3, a_4\}\}) &= -1, & w(\{a_1, a_3, a_4\}, \{\{a_2\}, \{a_1, a_3, a_4\}\}) &= 0, \\ w(\{a_3\}, \{\{a_3\}, \{a_1, a_2, a_4\}\}) &= 3, & w(\{a_1, a_2, a_4\}, \{\{a_3\}, \{a_1, a_2, a_4\}\}) &= 4, \\ w(\{a_4\}, \{\{a_4\}, \{a_1, a_2, a_3\}\}) &= 0, & w(\{a_1, a_2, a_3\}, \{\{a_4\}, \{a_1, a_2, a_3\}\}) &= 4, \\ w(\{a_1, a_2\}, \{\{a_1, a_2\}, \{a_3, a_4\}\}) &= 0, & w(\{a_3, a_4\}, \{\{a_1, a_2\}, \{a_3, a_4\}\}) &= 2, \\ w(\{a_1, a_3\}, \{\{a_1, a_3\}, \{a_2, a_4\}\}) &= 0, & w(\{a_2, a_4\}, \{\{a_1, a_3\}, \{a_2, a_4\}\}) &= 0, \\ w(\{a_1, a_4\}, \{\{a_1, a_4\}, \{a_2, a_3\}\}) &= 2, & w(\{a_2, a_3\}, \{\{a_1, a_4\}, \{a_2, a_3\}\}) &= 2, \\ w(\{a_1, a_2, a_3, a_4\}, \{\{a_1, a_2, a_3, a_4\}\}) &= 6. \end{aligned}$$

In this example, the optimal coalition structure CS^* is $\{\{a_1, a_2, a_4\}, \{a_3\}\}$, and its value is 7.

5.3 Partition Decision Trees

In this section, we introduce PDT, which were proposed by Skibski et al. (2015) [67].

In PDT, the game is represented by a set of PDT rules. A single PDT rule consists of a rooted directed tree, where non-leaf nodes are labeled with agent names, leaf nodes are labeled with payoff vectors, and edges are labeled with numbers that correspond to coalitions. Formally, PDT rule T is tuple $T = \langle V, E, x, f_V, f_E \rangle$, where

- (V, E) is a directed tree with root x ;
- $f_V : V \rightarrow N \cup \mathbb{R}^{|N|}$ is a label function for nodes;
- $f_E : E \rightarrow \{1, 2, \dots, |N|\}$ is a label function for edges, assuming that $f_V(v) \in N$ for every non-leaf v and $f_V(v) \in \mathbb{R}^N$ for every leaf v .

Given rule T , let $P(T)$ be the set of paths from the root to any leaf and let $last(p)$ denote the last leaf node in any $p \in P(T)$. Now, every path $p = (v_1, v_2, \dots, v_k) \in P(T)$ represents a partition of agents, where $f_E(v_i, v_{i+1})$ is the number of the coalition to which agent $f_V(v_i)$ belongs. Thus, for every such path,

- all non-leaf nodes on the path are labeled with different agents:

$$|\{f_V(v_1), f_V(v_2), \dots, f_V(v_{k-1})\}| = k - 1;$$

- for every path from x to v_i , the set of labels on the edges is the set of consecutive natural numbers beginning with 1. Thus, $f_E(v_1, v_2) = 1$, and a label of an edge is not bigger than the maximal label used earlier on this path plus one: $f_E(v_i, v_{i+1}) \leq \max_{1 \leq j < i} f_E(v_j, v_{j+1}) + 1$ for $1 \leq i < k$;
- the label of a leaf node is exactly the same size as the number of coalitions: $|f_V(v_k)| = \max_{1 \leq j < k} f_E(v_j, v_{j+1})$.

Coalition structure (partition) $CS \in \Pi(N)$ satisfies $p \in P(T)$ denoted by $CS \sim p$, if it covers the partition described by this path. Thus, for any two members of the same coalition that appear on the path, the labels of the outgoing edges are identical. Since paths describe different partitions, no more than one path exists in one PDT rule satisfied by partition CS . If $CS \sim p$, a mapping exists from the coalitions in partition CS to the set of labels of edges with zero: $g_p^{CS} : CS \rightarrow \{1, 2, \dots, \max_{1 \leq j < k} f_E(v_j, v_{j+1})\} \cup \{0\}$, where 0 is assigned to coalitions whose agents do not appear on the path.

From mapping g_p^{CS} for $CS \sim p$, for every coalition S embedded in coalition structure CS such that the agents in S appear on path p , there exists a unique value in a payoff node: $f_V(LAST(p))[g_p^{CS}(S)]$. For $\forall CS (CS \sim p), \forall S (S \in CS)$,

$$\delta_p^{CS}(S) = \begin{cases} f_V(LAST(p))[g_p^{CS}(S)] & \text{if } g_p^{CS}(S) > 0, \\ 0 & \text{otherwise} \end{cases} \quad (5.1)$$

The value of coalition S embedded in CS in game v^T , described by PDT rule T , is the value from the path satisfied by P :

$$w^T(S, CS) := \sum_{\substack{p \in \Pi(T), \\ P \sim p}} \delta_p^{CS}(S). \quad (5.2)$$

In every PDT rule, at most one path exists that is satisfied by particular coalition structure CS . If $\mathcal{T} = \{T_1, T_2, \dots\}$ is the set of PDT rules, then game $v^{\mathcal{T}}$, described by set of rules $\mathcal{T}$, is the sum of games described by each of the following rules:

$$w^{\mathcal{T}}(S, CS) := \sum_{T \in \mathcal{T}} w^T(S, CS) = \sum_{T \in \mathcal{T}} \sum_{\substack{p \in \Pi(T), \\ CS \sim p}} \delta_p^{CS}(S). \quad (5.3)$$

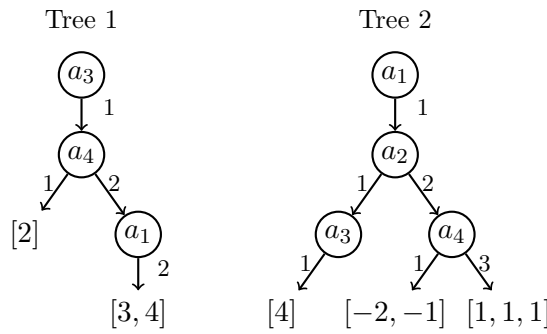


Figure 5.1. PDTs of coalitional game given in Example 19

Example 20. Figure 5.1 shows the PDTs of the game given in Example 19. The left-hand rule in Tree 1 indicates that when agents a_3 and a_4 belong to an identical coalition, 2 is added to the value of the coalition that includes a_3 and a_4 . The right-hand rule in Tree 1 indicates that when a_3 and a_4 are in different coalitions, but a_4 and a_1 are in an identical coalition, 3 is added to the value of the coalition that includes a_3 and 4 is added to the value of the coalition that includes a_4 and a_1 .

The value of coalition structure $W(CS)$ is calculated as the sum of the values of the compatible rules to CS . When CS is $\{\{a_1, a_2, a_3, a_4\}\}$, $W(CS)$ becomes $2 + 4 = 6$ by applying the rules indicated by the right-hand path in Tree 1 and the left-hand path in Tree 2.

PDT representations can be more concise than the existing representations. To represent the same game, weighted MC-nets [54] have a rule for every path of the PDT tree (whose size equals the size of the path), and embedded MC-nets need a rule for every value in the leaf of every path.

5.4 Branch-and-Bound Algorithm

In this section, we introduce a DFBnB algorithm for PDTs and another DFBnB algorithm for modified PDTs. For an optimization/maximization problem, we must carefully consider how to deal with negative rules. If no negative rule exists, we can straightforwardly solve a CSG problem.

5.4.1 Naive algorithm for positive rules

When all of the rules are positive, we seek as many compatible rules as possible. Thus, a CSG problem's dilemma is finding a set of compatible rules that maximizes the sum of the values by selecting at most a rule from each tree.

We show the outline of our DFBnB algorithm, which is guaranteed to obtain the optimal coalition structure:

- (1) We make a search tree by combining multiple PDTs. We first sort the PDTs by decreasing the average number of coalitions in their rules. In a search tree, a node indicates a rule. All of the rules exist in a single PDT at the same depth. Each rule in the i -th PDT appears at the i -th depth in the search tree. Then we sort the nodes by decreasing their value, i.e., the rule value at each depth. Here we add an empty node with 0-value as the last sibling node. Selecting the empty node indicates that no rule in the PDT is selected during the search.
- (2) We apply a DFBnB algorithm to a search tree. Here we restrict the search space with a heuristic function and also check the compatibility among rules on the path. Whenever we go down to a child node, we examine the compatibility between the obtained nodes on a path and the child node. If they are compatible, we expand one grandchild node. Otherwise, we move to a sibling node next to the child node. If there is no sibling node, we expand the sibling node next to the current node. If we reach a leaf node, the rules on the path imply a condition satisfied by the optimal coalition structure.
- (3) The obtained result does not imply a concrete coalition structure, since some agents might not appear in the set of rules. Such non-appeared agents do not affect the coalition's values in the result because the agents affecting the result must be examined in the search and appear in the set of rules. Therefore, we can add these non-appeared agents to any coalitions without changing their values. Currently, we add each of them to a coalition selected randomly to generate a concrete coalition structure.

Concerning compatibility between nodes, if we obtain $\{\{a_1, a_4\}, \{a_2\}, \{a_3\}\}$ at the i -th deepest node and the node of the $(i + 1)$ -st deepest node is $\{\{a_1, a_2, a_3\}\}$, we cannot combine $\{\{a_1, a_4\}, \{a_2\}, \{a_3\}\}$ with $\{\{a_1, a_2, a_3\}\}$.

5.4.2 Algorithm using modified PDTs

When a negative rule exists in PDTs, finding an optimal coalition structure is not straightforward. When a PDT has a negative rule, we need to examine whether the negative rule is incompatible with the other obtained rules. As a naive method, we first find an optimal set of rules that maximizes the sum of the values by ignoring the existence of the negative rules. If there exists a tree with negative rules in which a positive rule is not selected in an optimal set of rules, we examine whether the obtained rules are incompatible with every negative rule in the tree. Thus, we examine whether the set of obtained rules is compatible with all possible negations of the negative rules. This operation is very complicated, since its computational complexity depends on the number of agents included in the negative rules.

To solve the above difficulty, we modify the description of the PDTs and explicitly represent the rules with a value of 0 when a tree has a negative rule. If the original PDT is concisely represented, the negations of the negative rules are also concisely represented. For modified PDTs, we apply the same search algorithm proposed for the original PDTs in the previous subsection without adding an empty set in procedure (1).

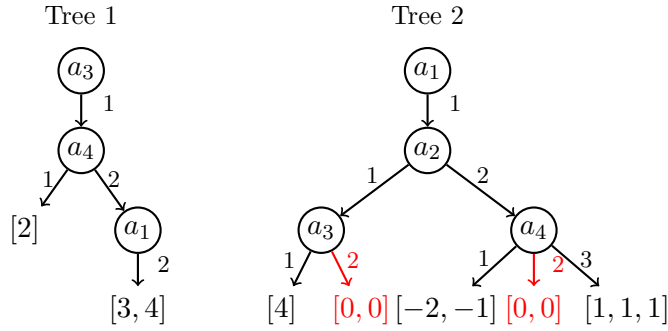


Figure 5.2. Modified PDTs of coalitional game given in Example 19

Example 21. Figure 5.2 shows our modified PDTs. In tree 2, since negative rules exist, we add two rules with the values of 0.

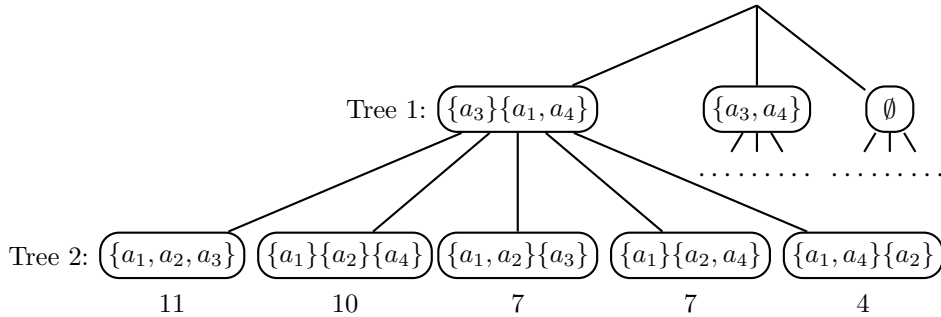


Figure 5.3. A search tree for the modified PDTs in Figure 5.2

Figure 5.3 shows a search tree generated by the modified PDTs in Figure 5.2. Since Tree 1 has only positive rules, we add an empty node that indicates a set of rules with a value of 0. In a search tree, a node indicates a rule. We sort the nodes by decreasing the node value. For this search tree, we apply a DFBnB algorithm. Whenever we go down to a child node, we examine the compatibility between the obtained nodes on a path and the child node. If they are compatible, we expand the grandchild node. Otherwise, we move to a sibling node next to the child node. If no sibling node exists, we expand the sibling node next to the node. If we reach the leaf node, the rules on the path imply a condition satisfied by the optimal coalition structure.

5.5 MaxSAT Encoding

To deal with a relation where two agents are in the same coalition, we introduce Boolean variable $C(i, j)$ for a pair of agents i and j where $i < j$. If i and j are in the same coalition, $C(i, j) = 1$, and otherwise, $C(i, j) = 0$. It is apparent that the relation is transitive, because any three agents, i , j , and k ($1 \leq i < j < k \leq n$), satisfy the following transitive laws: $\neg C(i, j) \vee \neg C(j, k) \vee C(i, k)$, $\neg C(i, j) \vee \neg C(i, k) \vee C(j, k)$, and $\neg C(i, k) \vee \neg C(j, k) \vee C(i, j)$, where n is the number of agents. The number of hard clauses for representing the transitive laws is $n \cdot (n - 1) \cdot (n - 2)/2$.

In MaxSAT encoding, we introduce a soft clause for a leaf in PDT rule T . Let $\pi \in P(T)$ be a path from the root to a leaf in T :

$$\pi : x_1 \xrightarrow{n_1} x_2 \xrightarrow{n_2} \dots x_{k-1} \xrightarrow{n_{k-1}} [r_1, r_2, \dots, r_m]$$

where x_i represents an agent, n_i is the coalition number to which x_i belongs, and $[r_1, r_2, \dots, r_m]$ represents a payoff vector. From π , we build set Ls^π of literals as a minimum set satisfying the following conditions (1) and (2), where $\text{FIRST}(n_i)$ denotes the first agent on π belonging to n_i :

- (1) When $\text{FIRST}(n_i) = x_i$, for all coalition numbers j in π satisfying $1 \leq j < n_i$, $\neg C(\text{FIRST}(j), x_i) \in Ls^\pi$ if $\text{FIRST}(j) < x_i$, $\neg C(x_i, \text{FIRST}(j)) \in Ls^\pi$ if $\text{FIRST}(j) > x_i$.
- (2) When $\text{FIRST}(n_i) \neq x_i$, $C(\text{FIRST}(n_i), x_i) \in Ls^\pi$ if $\text{FIRST}(n_i) < x_i$, $C(x_i, \text{FIRST}(n_i)) \in Ls^\pi$ if $\text{FIRST}(n_i) > x_i$.

(1) indicates that if x_i is the first agent belonging to n_i then x_i must not be in any coalition before n_i . (2) indicates that if x_i is not the first agent belonging to n_i then x_i must be in the same coalition as the first agent. Thus, Ls^π denotes a set of literals that hold in any partition satisfying π .

Example 22. Consider Tree 1 in Figure 5.1. For left path π_1 , $\text{FIRST}(1) = a_3$ and $Ls^{\pi_1} = \{C(a_3, a_4)\}$. For right path π_2 , $\text{FIRST}(1) = a_3$, $\text{FIRST}(2) = a_4$, and $Ls^{\pi_2} = \{\neg C(a_3, a_4), C(a_1, a_4)\}$.

Next, consider Tree 2. For left path π_3 , $\text{FIRST}(1) = a_1$, and $Ls^{\pi_3} = \{C(a_1, a_2), C(a_1, a_3)\}$. For middle path π_4 , $\text{FIRST}(1) = a_1$, $\text{FIRST}(2) = a_2$, and $Ls^{\pi_4} = \{\neg C(a_1, a_2), C(a_1, a_4)\}$. For right path π_5 , $\text{FIRST}(1) = a_1$, $\text{FIRST}(2) = a_2$, $\text{FIRST}(3) = a_4$, and $Ls^{\pi_5} = \{\neg C(a_1, a_2), \neg C(a_1, a_4), \neg C(a_2, a_4)\}$.

A soft clause and several hard clauses for path π are introduced with $Ls^\pi = \{L_1, L_2, \dots, L_l\}$ as follows where $R = \sum_{i=1}^m r_i$:

- (1) When $R > 0$, a soft clause (b_π, R) is introduced with a newly created variable b_π , and l hard clauses $\neg b_\pi \vee L_i$ ($1 \leq i \leq l$) are introduced.
- (2) When $R < 0$, a soft clause $(\neg L_1 \vee \neg L_2 \vee \dots \vee \neg L_l, -R)$ is introduced.
- (3) When $R = 0$, no clause is introduced.

(1) indicates that we obtain surplus R from a partition satisfying π in which each literal in Ls^π must hold. (2) indicates that we obtain surplus $-R$ from a partition that does not satisfy π . In other words, we do not obtain $-R$ from a partition that satisfies π . That is, we lose when the partition satisfies π . In this way, all negative values R s in PDT rules are negated. Therefore, after encoding, the final social surplus is $-W_{neg}$ larger than the original one, where W_{neg} is the sum of all negative values.

Example 23. Consider an identical example as Example 22. From the five paths, the following five soft clauses and eight hard clauses are introduced:

	Soft clause	Hard clause
π_1	$(b_{\pi_1}, 2)$	$\neg b_{\pi_1} \vee C(a_3, a_4)$
π_2	$(b_{\pi_2}, 7)$	$\neg b_{\pi_2} \vee \neg C(a_3, a_4), \neg b_{\pi_2} \vee C(a_1, a_4)$
π_3	$(b_{\pi_3}, 4)$	$\neg b_{\pi_3} \vee C(a_1, a_2), \neg b_{\pi_3} \vee C(a_1, a_3)$
π_4	$(C(a_1, a_2) \vee \neg C(a_1, a_4), 3)$	-
π_5	$(b_{\pi_5}, 3)$	$\neg b_{\pi_5} \vee \neg C(a_1, a_2), \neg b_{\pi_5} \vee \neg C(a_1, a_4),$ $\neg b_{\pi_5} \vee \neg C(a_2, a_4)$

5.6 Experimental Results

We experimentally evaluated the DFBnB performance for modified PDTs, called the MPDT algorithm and MaxSAT encoding on an Intel Xeon E5 Quad-Core 3.7 GHz processor (with 64 GB RAM) running a Mac OS X 10.10.5 with a 300-second timeout. MPDT is implemented in Java and runs on Java 1.8.0_40. We used our MaxSAT solver—QMAXSAT written in C++ and compiled with gcc 5.0.0.

We used randomly generated instances for our evaluation. Problem instances were generated in the following manner. First, make m subsets A_i ($i = 1, \dots, m$) of agents randomly to satisfy $\forall i \forall j (i \neq j) (A_i \not\subseteq A_j)$. Second, randomly generate l_i distinct partitions $p_1, \dots, p_{l_i}$ of A_i , and make PDT rule T_i to represent them. Thus, we obtain m PDT rules whose leaves remained unlabeled. Last, label each leaf of T_i with a payoff vector. Each value in the vector follows two distributions: normal or normally distribution coalition structures (NDCS). For normal distribution, we uniformly chose a value from $[0, |C|]$ at random, where C is the set of agents corresponding to the value. For NDCS, we chose a value based on the following distribution. Its average is $|C|$, and its standard deviation is $\sqrt{|C|}$. We convert the sum of the values whose corresponding coalitions are in the same leaf, to be negative with probability q .

The experiment was performed with the following parameter settings: number (#agents) of agents was 100 or 200 and number (#rules) of PDT rules was 10 or 20. The number (#leaves) of leaves in each rule was chosen from $[15, 25]$ or $[25, 35]$ so that their average was either 20 or 30, respectively. The number of agents in each rule was uniformly chosen from $[7, 13]$ at random. Probability q was 0, 0.2, or 0.4. Thus, we have 24 ($= 2 \times 2 \times 2 \times 3$) combinations of settings. For each setting, we generated 100 instances.

Tables 5.1 and 5.2 respectively show the experimental results with 100 and 200 agents. Notation “TnLl-m” in the first row denotes that the number of PDT rules

is n and the number of leaves in each rule is chosen from $[l, m]$. Five numbers are written in each cell. The bottom number surrounded by round brackets indicates the number of instances solved within 300 seconds. The number in the upper left indicates the average CPU time in seconds. The underlined number in the middle left indicates the standard deviation of the CPU time. The number surrounded by square brackets in the upper right indicates the number of nodes searched by MPDT or the number of times calling UP¹ by SAT-based MaxSAT. The underlined number surrounded by square brackets in the middle right indicates the standard deviation of the number of nodes for MPDT or the number of unit propagatings for MaxSAT.

The CSG problems become more difficult when $\#rules$ or $\#leaves$ increases in Tables 5.1 and 5.2. This is natural. If $\#rules$ or $\#leaves$ increase, the number of possible combinations of leaves in the PDT rules increases for MPDT, and the number of soft clauses increases for MaxSAT.

We also expected that the problems would become more difficult when $\#agents$ increase. But the experimental results indicate that this does not necessarily hold. Look at columns “T10L15-25” and “T10L25-35” in Table 5.1 and 5.2. The problems for $\#agents = 200$ are easier than those for $\#agents = 100$ for the following reason. There are 200 agents and 10 PDT rules in this setting. Since the number of agents in each rule is chosen between 7 and 13, approximately half of 200 agents (100 agents) are irrelevant for solving the CSG. This means that solving CSG for the setting is substantially the same as that for 100 agents. Furthermore, quite a few agents belong to more than one rule. This implies that a conflict between the rule combinations rarely happens. Thus, the problems are simplified.

As probability q increases, the CPU time increases for MPDT, but it decreases for MaxSAT. In MPDT, several 0-valued leaves are explicitly added to the PDT rule for its negative valued leaves. This increases the number of combinations to be examined and consumes CPU. On the other hand, for MaxSAT only one soft clause is generated for a negative valued leaf while a soft clause and several hard clauses are generated for a positive valued leaf. This means that MaxSAT solvers generally have to perform fewer inferences for negative valued leaves than those for the positive valued leaves. Thus, the CPU time for MaxSAT decreases as q increases.

Comparing MPDT and MaxSAT, MPDT outperforms MaxSAT when the problem only has positive values ($q = 0$), but MaxSAT outperforms MPDT, and otherwise ($q > 0$). For a PDT rule, the path from the root to a leaf is considered a partial coalition structure. MPDT takes these partial coalition structures one by one from each PDT rule and combines them to make a whole coalition structure. On the other hand, MaxSAT builds such a coalition structure from scratch, which explains why MPDT outperforms MaxSAT when $q = 0$.

Comparing the standard deviations of MPDT and MaxSAT, MPDT is unstable while MaxSAT is stable for solving the CSG. The DFBnB performance heavily depends on the order in which the rules are generally searched. The sorting works well on average but worse in some cases. These worse cases seem to increase the deviations.

Figure 5.4 and 5.5 compare two distributions for MPDT and MaxSAT, uniform and NDCS. Each point in the graphs represents a problem instance. Let its coordinate

¹ This corresponds to the space searched for by the SAT solver.

Table 5.1. {MPDT, MaxSAT} for {uniform, NDCS} distribution with 100 agents

Value distrib.	Solvers	Prob. of neg. value	T10L15-25	T10L25-35	T20L15-25	T20L25-35
uniform	MPDT	$q = 0.0$	1.26 [6.3E ⁵] 2.78 [1.4E ⁶] (100)	2.81 [1.5E ⁶] 7.32 [4.5E ⁶] (100)	119.30 [7.0E ⁷] 75.34 [4.4E ⁷] (35)	144.54 [8.4E ⁷] 48.76 [3.0E ⁷] (4)
		$q = 0.2$	33.46 [2.4E ⁷] 60.33 [4.3E ⁷] (84)	21.14 [1.4E ⁷] 40.38 [2.8E ⁷] (93)	– [–] – [–] (0)	– [–] – [–] (0)
		$q = 0.4$	40.14 [3.0E ⁷] 57.16 [4.4E ⁷] (87)	31.92 [2.2E ⁷] 53.24 [3.8E ⁷] (85)	– [–] – [–] (0)	– [–] – [–] (0)
	MaxSAT	$q = 0.0$	6.41 [2.1E ⁷] 2.42 [6.7E ⁶] (100)	15.51 [5.4E ⁷] 6.92 [2.0E ⁷] (100)	175.85 [3.3E ⁸] 65.90 [1.1E ⁸] (46)	255.18 [5.2E ⁸] 0 [0] (1)
		$q = 0.2$	5.11 [1.7E ⁷] 2.29 [5.8E ⁶] (100)	10.99 [3.9E ⁷] 4.66 [1.3E ⁷] (100)	189.38 [3.4E ⁸] 65.90 [1.0E ⁸] (45)	241.22 [4.9E ⁸] 32.18 [3.7E ⁷] (3)
		$q = 0.4$	3.36 [1.2E ⁷] 1.40 [3.7E ⁶] (100)	7.96 [2.9E ⁷] 2.94 [8.7E ⁶] (100)	161.98 [2.9E ⁸] 65.54 [1.0E ⁸] (80)	228.77 [4.6E ⁸] 54.19 [9.2E ⁷] (9)
NDCS	MPDT	$q = 0.0$	2.69 [1.5E ⁶] 4.67 [2.9E ⁶] (100)	7.50 [4.1E ⁶] 23.40 [1.4E ⁷] (100)	173.48 [1.1E ⁸] 84.50 [5.5E ⁷] (11)	– [–] – [–] (0)
		$q = 0.2$	50.13 [3.6E ⁷] 71.95 [5.3E ⁷] (79)	42.85 [2.9E ⁷] 69.36 [4.7E ⁷] (88)	– [–] – [–] (0)	– [–] – [–] (0)
		$q = 0.4$	33.46 [2.4E ⁷] 54.54 [3.9E ⁷] (83)	39.05 [2.6E ⁷] 63.25 [4.3E ⁷] (80)	– [–] – [–] (0)	– [–] – [–] (0)
	MaxSAT	$q = 0.0$	8.01 [2.6E ⁷] 2.79 [7.6E ⁶] (100)	19.36 [6.6E ⁷] 8.79 [2.4E ⁷] (100)	185.96 [3.3E ⁸] 52.70 [7.8E ⁷] (17)	– [–] – [–] (0)
		$q = 0.2$	6.17 [2.0E ⁷] 2.68 [7.2E ⁶] (100)	13.88 [4.9E ⁷] 5.57 [1.6E ⁷] (100)	206.98 [3.4E ⁸] 66.14 [9.1E ⁷] (24)	– [–] – [–] (0)
		$q = 0.4$	4.20 [1.4E ⁷] 1.74 [5.0E ⁶] (100)	9.70 [3.5E ⁷] 3.39 [9.7E ⁶] (100)	194.99 [3.2E ⁸] 55.73 [7.5E ⁷] (51)	269.95 [4.9E ⁸] 7.17 [4.3E ⁷] (2)

Table 5.2. {MPDT, MaxSAT} for {uniform, NDCS} distribution with 200 agents

Value distrib.	Solvers	Prob. of neg. value	T10L15-25	T10L25-35	T20L15-25	T20L25-35
uniform	MPDT	$q = 0.0$	0.71 [1.9E ⁵] 2.60 [7.7E ⁵] (100)	0.46 [1.1E ⁵] 2.03 [5.9E ⁵] (100)	173.46 [5.3E ⁷] 87.91 [2.8E ⁷] (5)	63.04 [1.8E ⁷] 9.20 [1.6E ⁶] (3)
		$q = 0.2$	4.46 [1.5E ⁶] 24.36 [8.5E ⁶] (100)	1.54 [5.0E ⁵] 5.88 [2.2E ⁶] (100)	– [–] – [–] (0)	– [–] – [–] (0)
		$q = 0.4$	3.66 [1.3E ⁶] 11.17 [4.1E ⁶] (100)	1.05 [3.2E ⁵] 3.32 [1.1E ⁶] (98)	– [–] – [–] (0)	– [–] – [–] (0)
	MaxSAT	$q = 0.0$	6.32 [1.3E ⁷] 2.51 [4.8E ⁶] (100)	11.21 [2.9E ⁷] 4.67 [9.8E ⁶] (100)	207.39 [2.7E ⁸] 62.27 [8.1E ⁷] (14)	241.87 [4.0E ⁸] 14.76 [1.9E ⁷] (3)
		$q = 0.2$	5.47 [1.0E ⁷] 1.94 [3.4E ⁶] (100)	8.84 [2.3E ⁷] 3.60 [7.7E ⁶] (100)	186.50 [2.3E ⁸] 64.67 [7.3E ⁷] (19)	283.64 [4.4E ⁸] 2.72 [7.0E ⁶] (3)
		$q = 0.4$	4.35 [7.7E ⁶] 2.00 [3.0E ⁶] (100)	6.71 [1.7E ⁷] 2.38 [5.1E ⁶] (100)	198.23 [2.3E ⁸] 59.42 [6.9E ⁷] (46)	213.27 [3.0E ⁸] 51.37 [6.1E ⁷] (12)
NDCS	MPDT	$q = 0.0$	0.78 [2.2E ⁵] 2.93 [9.5E ⁵] (100)	1.76 [5.4E ⁵] 7.83 [2.5E ⁶] (100)	282.63 [7.8E ⁷] 0.90 [3.5E ⁶] (2)	– [–] – [–] (0)
		$q = 0.2$	4.67 [1.6E ⁶] 24.78 [9.0E ⁶] (100)	1.82 [6.1E ⁵] 6.17 [2.4E ⁶] (100)	– [–] – [–] (0)	– [–] – [–] (0)
		$q = 0.4$	2.86 [9.4E ⁵] 12.42 [4.3E ⁶] (100)	2.30 [7.6E ⁵] 8.26 [3.1E ⁶] (99)	– [–] – [–] (0)	– [–] – [–] (0)
	MaxSAT	$q = 0.0$	6.87 [1.5E ⁷] 2.80 [5.6E ⁶] (100)	12.06 [3.2E ⁷] 5.60 [1.2E ⁷] (100)	262.42 [3.6E ⁸] 44.78 [4.2E ⁷] (6)	297.02 [5.2E ⁸] 0 [0] (1)
		$q = 0.2$	5.63 [1.1E ⁷] 2.23 [4.1E ⁶] (100)	9.17 [2.5E ⁷] 3.38 [7.7E ⁶] (100)	240.50 [3.0E ⁸] 46.53 [6.1E ⁷] (7)	280.82 [4.3E ⁸] 10.80 [1.5E ⁷] (2)
		$q = 0.4$	4.31 [8.2E ⁶] 1.56 [2.8E ⁶] (100)	7.11 [1.8E ⁷] 3.18 [7.0E ⁶] (100)	199.66 [2.3E ⁸] 65.25 [6.2E ⁷] (38)	210.04 [3.1E ⁸] 47.32 [5.9E ⁷] (8)

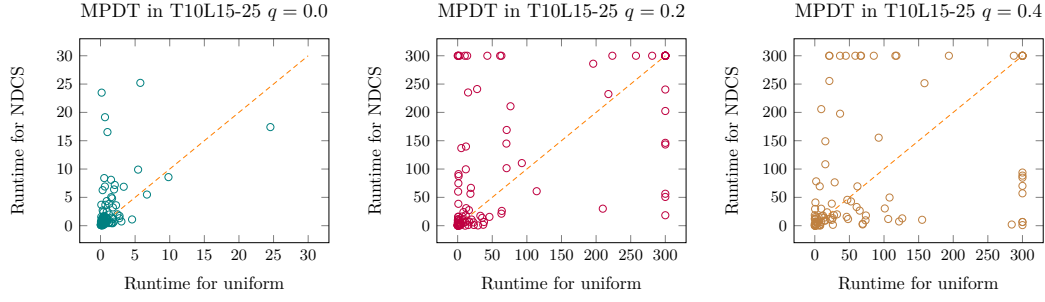


Figure 5.4. Uniform vs. NDCS in MPDT

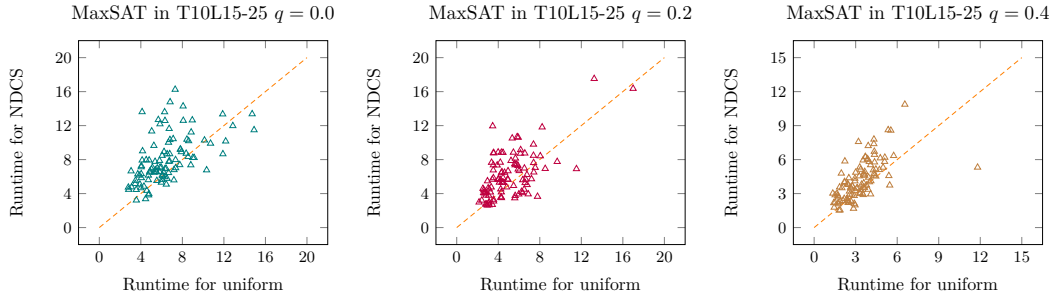


Figure 5.5. Uniform vs. NDCS in MaxSAT

be (x, y) . Recall that each leaf in the PDT rule is labeled with a payoff vector whose values follow a normal distribution or NDCS. x indicates the CPU time in seconds for an instance with a normal distribution while y indicates that with NDCS. Therefore, if a point is above the diagonal line, the corresponding instance with NDCS is more difficult than that with a normal distribution.

The instances become more difficult when their leaves are labeled with payoff vectors following NDCS as a whole. This coincides with the comparison between the corresponding average CPU times in Tables 5.1 and 5.2. The MPDT points are scattered while those of MaxSAT are gathered. This means that MaxSAT is stable for solving the CSG, as we have already have seen.

We studied a formulation using MIP. The formulation is basically identical as previous research [71]. Even though we preliminarily evaluated the formulation, we cannot solve any tiny instances that have only five PDT rules within ten minutes.

Chapter 6

Conclusions and Future Work

This dissertation has contributed to the research area of propositional satisfiability (SAT) and its optimization—maximum satisfiability (MaxSAT). We present two efficient implementations for SAT solving, which are embedded into our submitted solvers in SAT competitions. Then we focus on the pseudo-Boolean (PB) constraints encoding for MaxSAT and the transformation of coalition structure generation (CSG) problem to weighted partial MaxSAT (WPMS) problem. This chapter draws the conclusions from the research results obtained, and looks at some future work on related issues.

Efficient Implementations

This dissertation gives two original efficient implementations for improving the modern SAT solver. One focuses on the special strategy of pure literal elimination (PLE) which is regarded as another deduction engine of SAT algorithm, besides unit propagation (UP). We submitted our twin-engined solver with a tailored heuristics into the agile track of SAT competition 2016, and it achieved a comparable performance with other superior solvers in this track. The other enhances the management of learning and deleting conflict-driven clauses by a breadth-extended approach which is motivated by deriving more conflict-driven clauses in current truth assignments. We also submitted our solver with this implementation into SAT competition 2018, where it took second place of the random SAT track.

In our future work, for exhaustively seeking all detectable conflict-driven clauses, we need a major modification to basic SAT solver, especially in UP and learning phase. Moreover, we consider a hybrid implementation which combines PLE and breadth-extended approach for detecting and learning conflict-driven clauses.

PB constraints encoding for MaxSAT

It is known that the encoding of PB constraints into conjunctive normal form (CNF) can result very large formulas such that a SAT solver cannot handle the resulting CNF formula. Hence, new effective and compact encodings are crucial to the performance of MaxSAT solving. This dissertation discusses the encoding of PB constraints in the context of a linear search satisfiability-based solver for

MaxSAT, where a single PB constraint needs to be encoded. We presented several CNF encodings of PB constraints for WPMS and provided detailed proofs of the correctness for all our encodings. This work includes all our contributions to the development of encoding techniques in QMAXSAT. An n -level modulo totalizer (MTO), which is the newest of them, applies multiple modular arithmetic in a mixed radix base and only generates auxiliary variables for each unique combination of weights. Significantly, the n -level MTO always produces a polynomial-size CNF with respect to the number of input variables. Experimental results show that the n -level MTO generated a CNF whose size decreased more dramatically with totalizer and weighted totalizer and performed an improved speedup to solve the WPMS problem than other encodings.

Future work will introduce the n -level MTO to MINISAT⁺ for solving PB instances. For the critical problem that MTO destroys generalized arc consistency (GAC) property, we will also consider an algorithm for it that can detect inconsistency and maintain GAC by UP. On the other hand, we will investigate the unsatisfiable core-guided approach and intend to implement this approach into QMAXSAT.

MaxSAT encoding for the CSG problems

In this dissertation, we proposed two methods to solve the CSG problem represented by partition decision trees (PDTs). One is a depth-first branch-and-bound (DFBnB) algorithm for modified PDTs, called the MPDT algorithm, and the other is MaxSAT encoding. When PDT rules have negative valued leaves, the MPDT inserts several 0-valued nodes into the PDT rules, while MaxSAT negates both the constraints and the leaf values. In this respect, MaxSAT is probably superior to MPDT when the rules are negative. The experimental results support this superiority: MPDT outperforms MaxSAT when the problem has only positive values, but otherwise MaxSAT outperforms MPDT. Furthermore, MaxSAT is more stable than MPDT for solving the CSG problem.

In our future work, there are several avenues of this research that we hope are eventually explored:

- (1) compact the size of generated CNF for transitive laws in MaxSAT encoding,
- (2) perform more experiments to analyze the features of the methods,
- (3) improve the MPDT's stableness, and
- (4) further study the mixed integer programming for CSG.

Bibliography

- [1] AAVANI, A., MITCHELL, D. G., AND TERNOVSKA, E. New Encoding for Translating Pseudo-Boolean Constraints into SAT. In *Proceedings of the Tenth Symposium on Abstraction, Reformulation, and Approximation, SARA 2013, 11-12 July 2013, Leavenworth, Washington, USA*. AAAI Press (2013). Available from: <http://www.aaai.org/ocs/index.php/SARA/SARA13/paper/view/7212>.
- [2] ABÍO, I., NIEUWENHUIS, R., OLIVERAS, A., RODRÍGUEZ-CARBONELL, E., AND MAYER-EICHBERGER, V. A New Look at BDDs for Pseudo-Boolean Constraints. *J. Artif. Intell. Res.*, **45** (2012), 443. Available from: <https://doi.org/10.1613/jair.3653>, doi:10.1613/jair.3653.
- [3] AGGOUN, A. AND VAZACOPOULOS, A. *Solving Sports Scheduling and Timetabling Problems with Constraint Programming*, pp. 243–264. Springer Berlin Heidelberg (2004). ISBN 978-3-540-24734-0. Available from: https://doi.org/10.1007/978-3-540-24734-0_15, doi:10.1007/978-3-540-24734-0_15.
- [4] AGRAWAL, M., KAYAL, N., AND SAXENA, N. PRIMES is in P. *Annals of mathematics*, **160** (2004), 781. Available from: <http://www.jstor.org/stable/3597229>.
- [5] ANSÓTEGUI, C., BONET, M. L., AND LEVY, J. 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* (edited by O. Kullmann), pp. 427–440. Springer Berlin Heidelberg (2009). ISBN 978-3-642-02777-2. Available from: https://doi.org/10.1007/978-3-642-02777-2_39, doi:10.1007/978-3-642-02777-2_39.
- [6] ANSÓTEGUI, C., BONET, M. L., AND LEVY, J. 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*. AAAI Press (2010). Available from: <http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1774>.
- [7] ANSÓTEGUI, C., BONET, M. L., AND LEVY, J. SAT-based MaxSAT algorithms. *Artificial Intelligence*, **196** (2013), 77. Available from: <http://dx.doi.org/10.1016/j.artint.2013.01.002>, doi:10.1016/j.artint.2013.01.002.

- [8] ASÍN, R., NIEUWENHUIS, R., OLIVERAS, A., AND RODRÍGUEZ-CARBONELL, E. Cardinality Networks: a theoretical and empirical study. *Constraints*, **16** (2011), 195. Available from: <https://doi.org/10.1007/s10601-010-9105-0>, doi:10.1007/s10601-010-9105-0.
- [9] AUDEMARD, G., LAGNIEZ, J., AND SIMON, L. Improving Glucose for Incremental SAT Solving with Assumptions: Application to MUS Extraction. In *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings* (edited by M. Järvisalo and A. Van Gelder), pp. 309–317. Springer Berlin Heidelberg (2013). ISBN 978-3-642-39071-5. Available from: https://doi.org/10.1007/978-3-642-39071-5_23, doi:10.1007/978-3-642-39071-5_23.
- [10] AUDEMARD, G. AND SIMON, L. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 399–404. Morgan Kaufmann Publishers Inc. (2009). Available from: <http://ijcai.org/Proceedings/09/Papers/074.pdf>.
- [11] BAILLEUX, O. AND BOUFKHAD, Y. Efficient CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings* (edited by F. Rossi), pp. 108–122. Springer Berlin Heidelberg (2003). ISBN 978-3-540-45193-8. Available from: https://doi.org/10.1007/978-3-540-45193-8_8, doi:10.1007/978-3-540-45193-8_8.
- [12] BAILLEUX, O., BOUFKHAD, Y., AND ROUSSEL, O. A Translation of Pseudo Boolean Constraints to SAT. *JSAT*, **2** (2006), 191. Available from: <https://satassociation.org/jsat/index.php/jsat/article/view/25>.
- [13] BAILLEUX, O., BOUFKHAD, Y., AND ROUSSEL, O. New Encodings of Pseudo-Boolean Constraints into CNF. In *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings* (edited by O. Kullmann), pp. 181–194. Springer Berlin Heidelberg (2009). Available from: https://doi.org/10.1007/978-3-642-02777-2_19, doi:10.1007/978-3-642-02777-2_19.
- [14] BERRE, D. L. AND PARRAIN, A. The Sat4j library, release 2.2. *JSAT*, **7** (2010), 59. Available from: <https://satassociation.org/jsat/index.php/jsat/article/view/82>.
- [15] BIERE, A., HEULE, M., VAN MAAREN, H., AND WALSH, T. (eds.). *Handbook of Satisfiability*, vol. 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press (2009). ISBN 978-1-58603-929-5.
- [16] CHALKIADAKIS, G., ELKIND, E., AND WOOLDRIDGE, M. *Computational Aspects of Cooperative Game Theory*, vol. 5. Morgan & Claypool Publishers (2011). Available from: <https://doi.org/10.2200/S00355ED1V01Y201107AIM016>, doi:10.2200/S00355ED1V01Y201107AIM016.

- [17] CODISH, M., FEKETE, Y., FUHS, C., AND SCHNEIDER-KAMP, P. Optimal Base Encodings for Pseudo-Boolean Constraints. In *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings* (edited by P. A. Abdulla and K. R. M. Leino), pp. 189–204. Springer Berlin Heidelberg (2011). ISBN 978-3-642-19835-9. Available from: https://doi.org/10.1007/978-3-642-19835-9_16, doi:10.1007/978-3-642-19835-9_16.
- [18] CODISH, M. AND ZAZON-IVRY, M. Pairwise Cardinality Networks. In *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers* (edited by E. M. Clarke and A. Voronkov), pp. 154–172. Springer Berlin Heidelberg (2010). ISBN 978-3-642-17511-4. Available from: https://doi.org/10.1007/978-3-642-17511-4_10, doi:10.1007/978-3-642-17511-4_10.
- [19] COOK, S. A. The Complexity of Theorem-proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71*, pp. 151–158. ACM (1971). Available from: <http://doi.acm.org/10.1145/800157.805047>, doi:10.1145/800157.805047.
- [20] DANG, V. D., DASH, R. K., ROGERS, A., AND JENNINGS, N. R. Overlapping Coalition Formation for Efficient Data Fusion in Multi-Sensor Networks. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pp. 635–640. AAAI Press / The MIT Press (2006). Available from: <http://www.aaai.org/Library/AAAI/2006/aaai06-101.php>.
- [21] DAVIES, J. AND BACCHUS, F. Solving MAXSAT by Solving a Sequence of Simpler SAT Instances. In *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings* (edited by J. Lee), pp. 225–239. Springer Berlin Heidelberg (2011). ISBN 978-3-642-23786-7. Available from: https://doi.org/10.1007/978-3-642-23786-7_19, doi:10.1007/978-3-642-23786-7_19.
- [22] DAVIES, J. AND BACCHUS, F. Exploiting the Power of mip Solvers in maxsat. In *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings* (edited by M. Järvisalo and A. Van Gelder), pp. 166–181. Springer Berlin Heidelberg (2013). ISBN 978-3-642-39071-5. Available from: https://doi.org/10.1007/978-3-642-39071-5_13, doi:10.1007/978-3-642-39071-5_13.
- [23] DAVIES, J. AND BACCHUS, F. Postponing Optimization to Speed Up MAXSAT Solving. In *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings* (edited by C. Schulte), pp. 247–262. Springer Berlin Heidelberg

- (2013). ISBN 978-3-642-40627-0. Available from: https://doi.org/10.1007/978-3-642-40627-0_21, doi:10.1007/978-3-642-40627-0_21.
- [24] DAVIS, M., LOGEMANN, G., AND LOVELAND, D. A Machine Program for Theorem-Proving. *Commun. ACM*, **5** (1962), 394. Available from: <http://doi.acm.org/10.1145/368273.368557>, doi:10.1145/368273.368557.
- [25] DAVIS, M. AND PUTNAM, H. A Computing Procedure for Quantification Theory. *J. ACM*, **7** (1960), 201. Available from: <http://doi.acm.org/10.1145/321033.321034>, doi:10.1145/321033.321034.
- [26] EÉN, N. AND SÖRENSON, N. Translating Pseudo-Boolean Constraints into SAT. *JSAT*, **2** (2006), 1. Available from: <https://satassociation.org/jsat/index.php/jsat/article/view/18>.
- [27] FEI, L., MIZE, L., MOON, C., MULLEN, B., AND SINGHAL, S. Constraint analysis and debugging for multi-million instance SoC designs. In *11th International Symposium on Quality Electronic Design (ISQED)*, 22-24 March 2010, San Jose, CA, USA, pp. 422–427. IEEE (2010). Available from: <https://doi.org/10.1109/ISQED.2010.5450540>, doi:10.1109/ISQED.2010.5450540.
- [28] FEKETE, Y. AND CODISH, M. Simplifying Pseudo-Boolean Constraints in Residual Number Systems. 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* (edited by C. Sinz and U. Egly), pp. 351–366. Springer International Publishing (2014). ISBN 978-3-319-09284-3. Available from: https://doi.org/10.1007/978-3-319-09284-3_26, doi:10.1007/978-3-319-09284-3_26.
- [29] FU, Z. AND MALIK, S. 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* (edited by A. Biere and C. P. Gomes), pp. 252–265. Springer Berlin Heidelberg (2006). ISBN 978-3-540-37207-3. Available from: https://doi.org/10.1007/11814948_25, doi:10.1007/11814948_25.
- [30] GAREY, M. R. AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Books in mathematical series. W. H. Freeman (1979). ISBN 9780716710448.
- [31] GENT, I. P. Arc Consistency in SAT. In *Proceedings of the 15th European Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002*, pp. 121–125. IOS Press (2002). ISBN 978-1-58603-257-9. Available from: <https://ipg.host.cs.st-andrews.ac.uk/papers/ipgECAI.ps>.
- [32] GOLDBERG, E. AND NOVIKOV, Y. BerkMin: A fast and robust Sat-solver. *Discrete Applied Mathematics*, **155** (2007), 1549. SAT 2001, the Fourth International Symposium on the Theory and Applications of Satisfiability Testing. Available from: <http://dx.doi.org/10.1016/j.dam.2006.10.007>, doi:10.1016/j.dam.2006.10.007.

- [33] GOMES, C. P., SELMAN, B., AND KAUTZ, H. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pp. 431–437. AAAI Press / The MIT Press (1998). ISBN 0-262-51098-7. Available from: <http://www.aaai.org/Library/AAAI/1998/aaai98-061.php>.
- [34] HAYATA, S. AND HASEGAWA, R. Improvement in CNF Encoding of Cardinality Constraints for Weighted Partial MaxSAT. *SIG-FPAI, in Japanese*, **B4** (2015), 85. Available from: <http://id.nii.ac.jp/1004/00000592>.
- [35] HERAS, F., MORGADO, A., AND MARQUES-SILVA, J. 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*. AAAI Press (2011). Available from: <http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3713>.
- [36] HEYER, L. J. A Mathematical Optimization Problem in Bioinformatics. *PRIMUS*, **18** (2008), 101. Available from: <https://doi.org/10.1080/10511970701744992>, doi:10.1080/10511970701744992.
- [37] HÖLLDOBLER, S., MANTHEY, N., AND STEINKE, P. A Compact Encoding of Pseudo-Boolean Constraints into SAT. In *KI 2012: Advances in Artificial Intelligence - 35th Annual German Conference on AI, Saarbrücken, Germany, September 24-27, 2012. Proceedings* (edited by B. Glimm and A. Krüger), pp. 107–118. Springer Berlin Heidelberg (2012). ISBN 978-3-642-33347-7. Available from: https://doi.org/10.1007/978-3-642-33347-7_10, doi:10.1007/978-3-642-33347-7_10.
- [38] HOOKER, J. N. Planning and Scheduling by Logic-Based Benders Decomposition. *Operations Research*, **55** (2007), 588. Available from: <http://dx.doi.org/10.1287/opre.1060.0371>, doi:10.1287/opre.1060.0371.
- [39] IEONG, S. AND SHOHAM, Y. Marginal Contribution Nets: A Compact Representation Scheme for Coalitional Games. In *Proceedings of the 6th ACM Conference on Electronic Commerce, EC '05*, pp. 193–202. ACM (2005). ISBN 1-59593-049-3. Available from: <http://doi.acm.org/10.1145/1064009.1064030>, doi:10.1145/1064009.1064030.
- [40] JÄRVISALO, M., BERRE, D. L., ROUSSEL, O., AND SIMON, L. The International SAT Solver Competitions. *AI Magazine*, **33** (2012), 89. Available from: <https://doi.org/10.1609/aimag.v33i1.2395>, doi:10.1609/aimag.v33i1.2395.
- [41] JOHANNSSEN, J. The Complexity of Pure Literal Elimination. *Journal of Automated Reasoning*, **35** (2005), 89. Available from: <https://doi.org/10.1007/s10817-005-9008-8>, doi:10.1007/s10817-005-9008-8.
- [42] JOSHI, S., MARTINS, R., AND MANQUINHO, V. M. Generalized Totalizer Encoding for Pseudo-Boolean Constraints. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland*,

- August 31 - September 4, 2015, Proceedings* (edited by G. Pesant), pp. 200–209. Springer International Publishing (2015). Available from: https://doi.org/10.1007/978-3-319-23219-5_15, doi:10.1007/978-3-319-23219-5_15.
- [43] KELLERER, H., PFERSCHY, U., AND PISINGER, D. *Knapsack Problems*. Springer Berlin Heidelberg (2004). ISBN 978-3-540-40286-2. Available from: <https://doi.org/10.1007/978-3-540-24777-7>, doi:10.1007/978-3-540-24777-7.
- [44] KLEINBERG, J. M. AND TARDOS, É. *Algorithm Design*. Addison-Wesley (2006). ISBN 0-321-29535-8.
- [45] KOSHIMURA, M., ZHANG, T., FUJITA, H., AND HASEGAWA, R. QMaxSAT: A Partial Max-SAT Solver. *JSAT*, **8** (2012), 95. Available from: <https://satassociation.org/jsat/index.php/jsat/article/view/98>.
- [46] LARROSA, J. AND SCHIEX, T. Solving weighted CSP by maintaining arc consistency. *Artificial Intelligence*, **159** (2004), 1. Available from: <https://doi.org/10.1016/j.artint.2004.05.004>, doi:<https://doi.org/10.1016/j.artint.2004.05.004>.
- [47] LI, C. M. AND MANYÀ, F. MaxSAT, Hard and Soft Constraints. In Biere et al. [15], pp. 613–631. Available from: <https://doi.org/10.3233/978-1-58603-929-5-613>, doi:10.3233/978-1-58603-929-5-613.
- [48] LIAO, X., KOSHIMURA, M., FUJITA, H., AND HASEGAWA, R. Solving the Coalition Structure Generation Problem with MaxSAT. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, vol. 1, pp. 910–915. IEEE Computer Society (2012). Available from: <https://doi.org/10.1109/ICTAI.2012.127>, doi:10.1109/ICTAI.2012.127.
- [49] LIAO, X., KOSHIMURA, M., FUJITA, H., AND HASEGAWA, R. Extending MaxSAT to Solve the Coalition Structure Generation Problem with Externalities Based on Agent Relations. *IEICE Transactions*, **97-D** (2014), 1812. Available from: http://search.ieice.org/bin/summary.php?id=e97-d_7_1812.
- [50] LIAO, X., KOSHIMURA, M., FUJITA, H., AND HASEGAWA, R. MaxSAT Encoding for MC-Net-Based Coalition Structure Generation Problem with Externalities. *IEICE Transactions*, **97-D** (2014), 1781. Available from: http://search.ieice.org/bin/summary.php?id=e97-d_7_1781.
- [51] MANQUINHO, V. M., SILVA, J. P. M., AND PLANES, J. 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* (edited by O. Kullmann), pp. 495–508. Springer Berlin Heidelberg (2009). ISBN 978-3-642-02777-2. Available from: https://doi.org/10.1007/978-3-642-02777-2_45, doi:10.1007/978-3-642-02777-2_45.

- [52] MANTHEY, N., PHILIPP, T., AND STEINKE, P. A More Compact Translation of Pseudo-Boolean Constraints into CNF Such That Generalized Arc Consistency Is Maintained. In *KI 2014: Advances in Artificial Intelligence - 37th Annual German Conference on AI, Stuttgart, Germany, September 22-26, 2014. Proceedings* (edited by C. Lutz and M. Thielscher), pp. 123–134. Springer International Publishing (2014). ISBN 978-3-319-11206-0. Available from: https://doi.org/10.1007/978-3-319-11206-0_13, doi: [10.1007/978-3-319-11206-0_13](https://doi.org/10.1007/978-3-319-11206-0_13).
- [53] MICHALAK, T., MARCINIAK, D., SZAMOTULSKI, M., RAHWAN, T., WOOLDRIDGE, M., MCBURNEY, P., AND JENNINGS, N. R. A Logic-Based Representation for Coalitional Games with Externalities. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, pp. 125–132. AAMAS (2010). ISBN 978-0-9826571-1-9. Available from: <http://doi.acm.org/10.1145/1838206.1838224>, doi: [10.1145/1838206.1838224](https://doi.org/10.1145/1838206.1838224).
- [54] MICHALAK, T. P., RAHWAN, T., MARCINIAK, D., SZAMOTULSKI, M., AND JENNINGS, N. R. Computational aspects of extending the shapley value to coalitional games with externalities. In *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings*, pp. 197–202. IOS Press (2010). ISBN 978-1-60750-605-8. Available from: <http://www.booksonline.iospress.nl/Content/View.aspx?piid=17741>.
- [55] MOSKEWICZ, M. W., MADIGAN, C. F., ZHAO, Y., ZHANG, L., AND MALIK, S. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Annual Design Automation Conference, DAC '01*, pp. 530–535. ACM (2001). ISBN 1-58113-297-2. Available from: <http://doi.acm.org/10.1145/378239.379017>, doi: [10.1145/378239.379017](https://doi.org/10.1145/378239.379017).
- [56] OGAWA, T., LIU, Y., HASEGAWA, R., KOSHIMURA, M., AND FUJITA, H. Modulo Based CNF Encoding of Cardinality Constraints and Its Application to MaxSAT Solvers. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, November 4-6, 2013*, pp. 9–17. IEEE Computer Society (2013). Available from: <https://doi.org/10.1109/ICTAI.2013.13>, doi: [10.1109/ICTAI.2013.13](https://doi.org/10.1109/ICTAI.2013.13).
- [57] OHTA, N., CONITZER, V., ICHIMURA, R., SAKURAI, Y., IWASAKI, A., AND YOKOO, M. Coalition Structure Generation Utilizing Compact Characteristic Function Representations. In *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings* (edited by I. P. Gent), pp. 623–638. Springer Berlin Heidelberg (2009). ISBN 978-3-642-04244-7. Available from: https://doi.org/10.1007/978-3-642-04244-7_49, doi: [10.1007/978-3-642-04244-7_49](https://doi.org/10.1007/978-3-642-04244-7_49).
- [58] RAHWAN, T., MICHALAK, T. P., JENNINGS, N. R., WOOLDRIDGE, M., AND MCBURNEY, P. Coalition Structure Generation in Multi-Agent Systems with Positive and Negative Externalities. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California,*

- USA, July 11-17, 2009, pp. 257–263. Morgan Kaufmann Publishers Inc. (2009). Available from: <http://ijcai.org/Proceedings/09/Papers/052.pdf>.
- [59] ROUSSEL, O. AND MANQUINHO, V. M. Pseudo-Boolean and Cardinality Constraints. In Biere et al. [15], pp. 695–733. Available from: <https://doi.org/10.3233/978-1-58603-929-5-695>, doi:10.3233/978-1-58603-929-5-695.
- [60] SANDHOLM, T., LARSON, K., ANDERSSON, M., SHEHORY, O., AND TOHMÉ, F. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, **111** (1999), 209. Available from: [https://doi.org/10.1016/S0004-3702\(99\)00036-3](https://doi.org/10.1016/S0004-3702(99)00036-3), doi:10.1016/S0004-3702(99)00036-3.
- [61] SANDHOLM, T. AND LESSER, V. R. Coalitions among computationally bounded agents. *Artificial Intelligence*, **94** (1997), 99. Economic Principles of Multi-Agent Systems. Available from: [https://doi.org/10.1016/S0004-3702\(97\)00030-1](https://doi.org/10.1016/S0004-3702(97)00030-1), doi:10.1016/S0004-3702(97)00030-1.
- [62] SILVA, J. P. M. *Search Algorithms for Satisfiability Problems in Combinational Switching Circuits*. Ph.D. thesis, University of Michigan, Ann Arbor, MI, USA (1995). UMI: AAI9542904. Available from: <https://deepblue.lib.umich.edu/handle/2027.42/104664>.
- [63] SILVA, J. P. M. AND LYNCE, I. Towards Robust CNF Encodings of Cardinality Constraints. In *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings* (edited by C. Bessière), pp. 483–497. Springer Berlin Heidelberg (2007). ISBN 978-3-540-74970-7. Available from: https://doi.org/10.1007/978-3-540-74970-7_35, doi:10.1007/978-3-540-74970-7_35.
- [64] SILVA, J. P. M. AND SAKALLAH, K. A. Dynamic Search-space Pruning Techniques in Path Sensitization. In *Proceedings of the 31st Annual Design Automation Conference, DAC '94*, pp. 705–711. ACM (1994). ISBN 0-89791-653-0. Available from: <http://doi.acm.org/10.1145/196244.196621>, doi:10.1145/196244.196621.
- [65] SILVA, J. P. M. AND SAKALLAH, K. A. GRASP—A New Search Algorithm for Satisfiability. In *Proceedings of International Conference on Computer Aided Design*, pp. 220–227. IEEE Computer Society (1996). Available from: <https://doi.org/10.1109/ICCAD.1996.569607>, doi:10.1109/ICCAD.1996.569607.
- [66] SINZ, C. 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* (edited by P. van Beek), pp. 827–831. Springer Berlin Heidelberg (2005). ISBN 978-3-540-32050-0. Available from: https://doi.org/10.1007/11564751_73, doi:10.1007/11564751_73.
- [67] SKIBSKI, O., MICHALAK, T. P., SAKURAI, Y., WOOLDRIDGE, M., AND YOKOO, M. A Graphical Representation for Games in Partition Function Form. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*,

- January 25-30, 2015, Austin, Texas, USA., pp. 1036–1042. AAAI Press / The MIT Press (2015). Available from: <http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9624>.
- [68] TARJAN, R. E. Finding Dominators in Directed Graphs. *SIAM J. Comput.*, **3** (1974), 62. Available from: <https://doi.org/10.1137/0203006>, doi:10.1137/0203006.
- [69] THRALL, R. M. AND LUCAS, W. F. N-Person Games in Partition Function Form. *Naval Research Logistics Quarterly*, **10** (1963), 281. Available from: <https://doi.org/10.1002/nav.3800100126>, doi:10.1002/nav.3800100126.
- [70] TSEITIN, G. S. On the Complexity of Derivation in Propositional Calculus. In *Automation of Reasoning: Classical Papers on Computational Logic 1967-1970*, vol. 2, pp. 466–483. Springer (1983). ISBN 978-3-642-81955-1. Available from: https://doi.org/10.1007/978-3-642-81955-1_28, doi:10.1007/978-3-642-81955-1_28.
- [71] UEDA, S., HASEGAWA, T., HASHIMOTO, N., OHTA, N., IWASAKI, A., AND YOKOO, M. Handling Negative Value Rules in MC-net-based Coalition Structure Generation. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pp. 795–804. AAMAS (2012). ISBN 0-9817381-2-5, 978-0-9817381-2-3. Available from: http://ifaamas.org/Proceedings/aamas2012/papers/1E_5.pdf.
- [72] WARNERS, J. P. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, **68** (1998), 63. Available from: [https://doi.org/10.1016/S0020-0190\(98\)00144-6](https://doi.org/10.1016/S0020-0190(98)00144-6), doi:10.1016/S0020-0190(98)00144-6.
- [73] WOOD, R. G. AND RUTENBAR, R. A. FPGA Routing and Routability Estimation via Boolean Satisfiability. In *Proceedings of the 1997 ACM Fifth International Symposium on Field-programmable Gate Arrays, FPGA '97*, pp. 119–125. ACM (1997). ISBN 0-89791-801-0. Available from: <http://doi.acm.org/10.1145/258305.258322>, doi:10.1145/258305.258322.
- [74] YEH, D. Y. A Dynamic Programming Approach to the Complete Set Partitioning Problem. *BIT*, **26** (1986), 467. Available from: <http://dx.doi.org/10.1007/BF01935053>, doi:10.1007/BF01935053.
- [75] ZHA, A., KOSHIMURA, M., AND FUJITA, H. A Hybrid Encoding of Pseudo-Boolean Constraints into CNF. In *Conference on Technologies and Applications of Artificial Intelligence, TAAI 2017, Taipei, Taiwan, December 1-3, 2017*, pp. 9–12. IEEE Computer Society (2017). Available from: <https://doi.org/10.1109/TAAI.2017.15>, doi:10.1109/TAAI.2017.15.
- [76] ZHA, A., NOMOTO, K., UEDA, S., KOSHIMURA, M., SAKURAI, Y., AND YOKOO, M. Coalition Structure Generation for Partition Function Games Utilizing a Concise Graphical Representation. In *PRIMA 2017: Principles and Prac-*

- tice of Multi-Agent Systems - 20th International Conference, Nice, France, October 30 - November 3, 2017, Proceedings* (edited by B. An, A. Bazzan, J. Leite, S. Villata, and L. van der Torre), vol. 10621 of *Lecture Notes in Computer Science*, pp. 143–159. Springer International Publishing (2017). ISBN 978-3-319-69131-2. Available from: https://doi.org/10.1007/978-3-319-69131-2_9, doi:10.1007/978-3-319-69131-2_9.
- [77] ZHA, A., UEMURA, N., KOSHIMURA, M., AND FUJITA, H. Mixed Radix Weight Totalizer Encoding for Pseudo-Boolean Constraints. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence, Boston, MA, USA, November 6-8, 2017*, pp. 868–875. IEEE Computer Society (2017). Available from: <https://doi.org/10.1109/ICTAI.2017.00135>, doi:10.1109/ICTAI.2017.00135.
- [78] ZHANG, L. *Searching for Truth: Techniques for Satisfiability of Boolean Formulas*. Ph.D. thesis, Princeton University, Princeton, NJ, USA (2003). UMI: AAI3102236. Available from: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/lintaos-thesis_lintao_zhang.pdf.

Publications by the Author

- [1] ZHA, A., KOSHIMURA, M., AND FUJITA, H. A Hybrid Encoding of Pseudo-Boolean Constraints into CNF. In *Conference on Technologies and Applications of Artificial Intelligence, TAAI 2017, Taipei, Taiwan, December 1-3, 2017*, pp. 9–12. IEEE Computer Society (2017). Available from: <https://doi.org/10.1109/TAAI.2017.15>, doi:10.1109/TAAI.2017.15.
- [2] ZHA, A., NOMOTO, K., UEDA, S., KOSHIMURA, M., SAKURAI, Y., AND YOKOO, M. Coalition Structure Generation for Partition Function Games Utilizing a Concise Graphical Representation. In *PRIMA 2017: Principles and Practice of Multi-Agent Systems - 20th International Conference, Nice, France, October 30 - November 3, 2017, Proceedings* (edited by B. An, A. Bazzan, J. Leite, S. Villata, and L. van der Torre), vol. 10621 of *Lecture Notes in Computer Science*, pp. 143–159. Springer International Publishing (2017). ISBN 978-3-319-69131-2. Available from: https://doi.org/10.1007/978-3-319-69131-2_9, doi:10.1007/978-3-319-69131-2_9.
- [3] ZHA, A., UEMURA, N., KOSHIMURA, M., AND FUJITA, H. Mixed Radix Weight Totalizer Encoding for Pseudo-Boolean Constraints. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence, Boston, MA, USA, November 6-8, 2017*, pp. 868–875. IEEE Computer Society (2017). Available from: <https://doi.org/10.1109/ICTAI.2017.00135>, doi:10.1109/ICTAI.2017.00135.

Subject Index

- GLUCOSE 3.0, 28, 33, 54
 - literal block distance, 31
- QMAXSAT, 37, 38, 54, 57, 69, 76
 - κ -simplification, 42, 44, 45, 47, 52
 - comparator, 42, 44, 47, 51
 - modulo totalizer encoding, 39, 45–47, 49–52, 54, 76
 - totalizer encoding, 38–45, 54, 76
 - weighted totalizer encoding, 39, 43–46, 54, 76
- Boolean algebra, 2, 5
 - equisatisfiable, 5, 15
 - implication, 3, 16, 19, 22, 31
- breadth-extended approach, 75
- central processing unit, 3
- circuit, 3
- clause deletion, 23, 29, 33, 75
 - learnt clause database, 32, 33, 75
- clause learning, 21, 29, 31, 32, 75
 - antecedent, 19
 - asserting clause, 20, 21
 - conflict analysis, 19, 22, 23, 29, 31, 32
 - conflict set, 20
 - conflict-driven clause, 15, 19, 20, 28, 31, 32, 75
 - detectable conflict, 31, 32
 - dominator, 21
 - implication graph, 19, 20, 31
 - unique implication point, 20, 31
- coalitional game, 61, 62
 - agent-based approach, 62
 - characteristic function game, 61, 62
- coalition structure generation, 61–63, 65, 66, 70, 73, 75, 76
- compatibility, 65–67
- embedded marginal contribution nets, 62, 65
- externality, 61, 62
- partition decision tree, 61, 62, 64–70, 73, 76
- partition function game, 61, 62
- relation-based approach, 62
- transitive law, 68, 76
- combinatorics
 - combinatorial optimization, 36, 38, 54
 - combinatorial problem, 5, 8
 - weighted combination, 39, 43–45, 76
- complexity theory, 9, 10, 62
 - asymmetric, 11
 - asymptotic, 9, 43
 - complement, 11
 - completeness, 12
 - complexity class, 11
 - efficient, 9, 11
 - polynomial reduction, 12
 - polynomial size, 35, 43, 47, 52, 58, 76
 - polynomial time, 10, 12, 37, 62
 - symmetric, 11
 - verifier, 11
- conjunctive normal form, 4, 8, 16, 25, 32, 35, 37–42, 46, 47, 50, 51, 54, 58, 75, 76
- auxiliary variable, 37, 38, 41, 43, 45, 47, 52–54, 58, 76
- bijection, 43

- Boolean variable, 4, 7, 15, 30, 37, 39, 42–47, 49, 50, 52, 68
- clause, 4, 19, 25, 30, 40, 42, 47, 52, 54, 58
- encoding, 13, 35, 38, 39, 42, 54, 58, 75
- falsify, 8, 35
- hard clause, 35, 36, 68–70
- literal, 4, 19, 25, 30
- negation, 4, 17, 20
- pure literal, 26, 27
- satisfiable, 8, 30, 35, 37, 38, 42, 47, 52, 68, 69
- soft clause, 35–38, 68–70
- subsumption, 16, 17, 33
- Tseitin encoding, 5
- unit clause, 17, 19, 22, 30, 31, 42
- unsatisfiable, 8, 30, 37, 38
- unsatisfiable core, 54, 58, 76
- variable polarity, 17, 26, 28
- variables partition, 39, 43
- weight, 35, 41, 43–45, 49, 52, 58
- decision problem, 4, 8, 10
- deduction engine, 26–28, 75
 - pure literal detection, 27
 - pure literal elimination, 26, 75
 - pure literal rule, 26
 - unit propagation, 16, 19, 26, 29, 31, 32, 40, 58, 70, 75
- dynamic adjustment approach, 28
 - execution frequency, 28
- efficient implementation, 25, 29, 75
 - breadth-extended approach, 31
 - dummy assignment, 32
 - vector, 25, 27, 31
- integer interval, 39–41, 45–47, 49, 50
 - adder, 40, 41, 43, 44, 46, 47, 49, 52
 - unary representation, 39, 41, 43
 - weighted unary representation, 43, 45, 49
- lazy data structure, 23, 29, 30
 - clause' reference, 30, 31
 - two watched literals, 30, 31
- watch list, 31, 32
- watched literal data structure, 29–32
- logical system, 1
 - completeness, 1, 23
 - consistency, 1, 18
 - derivation, 17, 21, 31, 40, 75
 - inference, 1, 15
 - logical truth, 2
 - necessity, 2
 - possibility, 2
 - soundness, 15, 23
 - tautology, 2, 3, 41
 - validity, 1
- mixed integer programming, 54, 58, 62, 73, 76
- modular arithmetic, 35, 45, 76
 - base selection, 35, 52, 53
 - carry, 46, 50, 52
 - digit, 45–47, 49–52
 - mixed radix base, 39, 49, 52, 53, 76
 - modulus, 45, 47, 49, 52
 - multi-level modulo, 49–52
 - rate of divisibility, 53
 - single-level modulo, 45–47
- multi-agent system, 61
- propositional logic, 2, 5
 - 3-SAT, 6, 8, 13
 - circuit satisfiability, 3, 8, 13
 - encoding, 4, 62, 68, 69, 75, 76
 - existential quantification, 15
 - resolution, 15, 23
 - truth assignment, 3
- propositional satisfiability, 8, 13, 35, 75
 - arc consistency, 40, 44, 58, 76
 - maximum satisfiability, 8, 35, 38, 70, 73, 75
 - optimal assignment, 36, 37
 - optimal cost, 36, 37
 - pseudo-Boolean constraint, 8, 37–39, 41–44, 47, 50, 51, 54, 58, 75
 - SAT-based approach, 9, 37, 70

- truth assignment, 8, 16, 26, 31, 35, 37–42, 45, 47, 49, 50, 52, 75
- random access memory, 15
- semantics, 1, 3
 - algebraic structure, 1
 - model theory, 1
- syntax, 1
 - axiom system, 1
 - formula, 2
 - proof theory, 1
 - satisfiable, 2, 4, 62, 64–67
 - theorem-hood, 2
 - unsatisfiable, 2
- systematic search, 15, 16
 - assertion level, 21
 - backtracking, 17, 21, 28, 31, 33
 - branch-and-bound algorithm, 37, 62, 65–67, 69, 70, 76
 - brute-force search, 16, 17
 - complete binary tree, 16
 - contradiction, 17, 20, 21, 31
 - decision assignment, 16, 20, 22, 27
 - decision heuristics, 17, 31
 - decision level, 16, 18, 20, 28, 30, 31
 - decision stack, 16
 - decision variable, 17, 27
 - depth-first-search, 16, 17, 62, 65–67, 69, 70, 76
 - derivation, 19
 - dummy decision assignment, 28
 - search space, 10, 20, 28, 40, 66
 - search tree, 16, 17, 30, 66, 67
 - splitting rule, 15