Studies on Applying Incremental SAT Solving to Optimization and Enumeration Problems

Takehide Soh

DOCTOR OF PHILOSOPHY

Department of Informatics,
School of Multidisciplinary Sciences

The Graduate University for Advanced Studies

2011

Acknowledgements

I would like to thank my supervisor Professor Katsumi Inoue for his guidance throughout my graduate studies. I have studied the research topics around SAT technologies in his laboratory since I was undergraduate student. Without his generous support and patience, I could never complete this thesis. I have learned not only how to do research but also my attitude towards research. His passion and dedication to research will always be a model for me to follow.

I am very grateful to Professor Naoyuki Tamura, Professor Ken Sato, Professor Hiroshi Hosobe and Professor Takeaki Uno for their fruitful comments. Their advice on the research topics encouraged me to write this thesis. I would like to present my sincere thanks to all members of CSPSAT project for their worthful comments for my work. In particular, I would like to thank Professor Koji Iwanuma, Professor Mutsunori Banbara and Professor Hidetomo Nabeshima for their constant and kind support.

Motivation for this thesis is also inspired from discussions in National Institute of Informatics (NII) and other institutions I visited. I would like to thank Professor Andrei Doncescu, with whom I learned the interest of systems biology. I would also like to thank Professor Silvano Martello for his worthful comments for solving optimization problems. I would like to thank Dr. Oliver Ray for his kind support. Discussions with him greatly inspired me. I would like to thank Dr. Ateet Bhalla for his worthful comments. I would also like to express my appreciation for Professor Taisuke Sato and

Professor Chiaki Sakama for their fruitful comments for my work, Professor Laurent Simon for valuable discussions in le Laboratoire de Recherche en Informatique, Professor Thomas Eiter and Professor Alexander Bochman for their productive discussions in NII. I was also inspired by discussions with many colleagues. I would like to thank Dr. Lena Wiese, Dr. Petr Buryan, Gabriel Synnaeve, Matej Holec, James Ray Wagner, Domenico Corapi, Johannes Oetsch, Jiefei Ma, Tomoya Tanjo, Masakazu Ishihata and others. I would also like to thank the anonymous referees of several conferences and workshops for their feedback on parts of this thesis.

I gladly acknowledge the financial support of NII as well as the Japan Science and Technology Agency (JST), le Centre National de la Recherche Scientifique (CNRS) and the Japan Society for the Promotion of Science (JSPS) under 2007-2009 JST-CNRS Strategic Japanese-French Cooperative Program, 2008-2011 JSPS Grant-in-Aid Scientic Research (A) (No. 20240016), 2008-2011 JSPS Grant-in-Aid Scientic Research (A) (No. 20240003), and 2010-2011 JSPS fellowship program. I also acknowledge the financial support of the Japan Student Services Organization and the Korean Scholarship Foundation.

I would like to express my great appreciation for Dr. Yoshitaka Yamamoto, Dr. Gauvain Bourgne and Hiroaki Watanabe, and the secretaries of our laboratory and the staff of the Graduate University for Advanced Studies for their kind help and constant encouragement.

Last but not least, I would like to thank my parents Tameichi and Tomoko, my sisters Masumi and Tomomi, and my brother Noritaka for their understanding. Without their continuous support, I could not overcome tough times in my life.

Abstract

Optimization and enumeration problems have been actively studied. There are not only academic interests but also real world applications as many researches have been done for industrial purpose. However, despite the recent advancements in computer technology, there are still difficult problems to solve. To break this situation, we focus on technologies in solving propositional satisfiability (SAT). Although SAT technologies are not so focused as a method for optimization and enumeration problems, the recent progress of SAT technologies is so tremendous that it can be expected to become a potential approach.

In this thesis, we study a method called incremental SAT solving. It incrementally computes the satisfiability of a sub-problem obtained from a given original problem until a given goal condition is satisfied. We review how previous approaches utilizing SAT technologies are explained by this solving method. Following that, we also review several applications of it and how to utilize learned clauses for acceleration. The main contribution of this thesis is applying this incremental SAT solving to the following optimization and enumeration problems.

We apply incremental SAT solving to an optimization problem, the two-dimensional strip packing problem (2SPP). In this problem, we are given a set of rectangles and one large rectangle called a strip. The goal of the problem is to pack all rectangles without overlapping, into the strip by minimizing the overall height of the packing. Although the 2SPP has been studied in operations research, some instances are still hard to

solve. Our method solves the 2SPP by translating it into SAT problems through a propositional encoding called order encoding. The size of translated SAT problems tends to be large; thus, we apply several techniques to reduce the search space by symmetry breaking and positional relations of rectangles. Besides that, to compute the minimum height of a 2SPP, we apply incremental SAT solving with reusing learned clauses. For evaluation, we make comparisons with a constraint satisfaction solver and ad-hoc methods of 2SPP on 38 instances obtained from the literature.

We then apply incremental SAT solving to the minimal active pathway finding problem that we propose to analyze metabolic pathways. In systems biology, identifying vital functions like glycolysis from a given metabolic pathway is important for better understanding of living organisms. The goal of the problem is to identifying such functional reaction sets in a given metabolic pathways. More specifically, we focus on enumerating minimal active pathways producing target metabolites from source metabolites. We translate laws of biochemical reactions into propositional formulas and apply incremental SAT solving to solve the problem. An advantage of our method is that each solution satisfies qualitative laws of biochemical reactions. Moreover, we can calculate such solutions for a cellular scale metabolic pathway within a few seconds. For evaluation, we apply it to a whole *Escherichia coli* metabolic pathway and make comparisons with previous approaches.

In this thesis, we mainly investigate applications of SAT technologies. Apparently, it is simple and seems difficult to apply it to real world problems. However, it has remarkable potential to be core solvers for problems such as optimization and enumeration problems as we show throughout this thesis.

Contents

1 Introduction **1**

1.1 Background 1

1.1.1 Overview of SAT 2

1.1.2 SAT for Problem Solving 4

1.1.3 Incremental SAT Solving 5

1.1.4 Further Applications 6

1.2 Contributions 7

1.3 Thesis Organization 9

1.4 Publications 9

2 Propositional Logic and SAT **12**

2.1 Propositional Formulas and its Satisfiability 12

2.1.1 Syntax 12

2.1.2 Semantics 14

2.1.3 SAT Problem 15

2.2 Modern SAT Solvers and Techniques 15

2.2.1 DPLL Procedure 16

2.2.2 CDCL Solvers and Techniques 18

2.2.3 Clause Learning 21

2.2.4 Other Important Techniques 23

2.3 Model Generation by SAT Solvers 26

3 Incremental Satisfiability Solving **31**

3.1 Architecture of the Solving Method based on SAT Technologies 31

3.2 Incremental Satisfiability Solving 33

3.3 Stepwise Approach to Optimization Problems 35

3.4 Stepwise Enumeration 42

3.5 Reusing Learned Clauses 43

3.6 Related Work 46

3.7 Summary 48

4 Two-dimensional Strip Packing Problem **49**

4.1 Introduction 49

4.2 Preliminaries 52

4.2.1 2SPP and 2OPP 52

4.2.2 Order Encoding 54

4.3 Encoding from 2OPP into SAT 56

4.4 Solving 2SPP by Incremental SAT Solving 58

4.4.1 Searching for Optimum Height of 2SPP 59

4.4.2 Reusing Learned Clauses 61

4.4.3 Reduction Techniques 62

4.5 Experimental Results 64

4.5.1 Benchmarks and Environments 64

4.5.2 Results for 2OPP Instances 65

4.5.3 Results for 2SPP Instances 70

4.5.4 Comparison with Previous 2SPP Method 72

4.6 Discussion 74

4.7 Related Work 76

4.8 Summary 77

5 Minimal Active Pathway Finding Problem 79

5.1 Introduction 79

5.2 Minimal Active Pathway Finding Problem 81

5.3 Incremental SAT Solving for MAPF 85

5.3.1 Overview 85

5.3.2 Reduce Reactions for Encoding 86

5.3.3 Encoding 88

5.3.4 Treating Reversible Reactions and Cycles 92

5.3.5 Biological Applications 93

5.4 Experiments and Results 95

5.4.1 Comparison with Previous Methods 95

5.4.2 Evaluation on the Whole *E. coli* Metabolic Pathway from EcoCyc 97

5.5 Related Work 101

5.6 Summary 103

6 Conclusions and Future Work 104

6.1 Incremental SAT Solving and Applications 104

6.2 Future Work 106

6.2.1 Encoding Methods and Other Representation 106

6.2.2 Accelerating Incremental SAT Solving 107

6.2.3 Further Applications and Extensions 107

List of Figures

2.1 DPLL Algorithm 17
2.2 Algorithm of CDCL Solvers 20
2.3 Example of an Implication Graph 23
2.4 Model Generation 27
2.5 Minimal Model Generation Procedure [86] 29

3.1 Solving Method based on SAT Technologies 32
3.2 Incremental SAT Solving 33
3.3 Computation Time around the Optimal Value on 2SPP 38
3.4 Bisection Method 41
3.5 Bisection Method [74] 42
3.6 Abstracted Conflict Graph [104] 44

4.1 Example of a 2SPP 56
4.2 Example of a SAT-encoded 2OPP 58
4.3 Size of the Encoded SAT problem 59
4.4 Framework for Solving 2OPP 60
4.5 Symmetry Breaking and Reduction Techniques 63
4.6 Computed Placement of HT08 71
4.7 Representation of Conflicts 75

5.1 The Architecture for Analyzing Metabolic Pathways 86

5.2 Procedure for Reduction 87
5.3 Example of Reversible Reactions 92
5.4 Glycolysis Active Pathway on a Whole *E. coli* Pathway 98
5.5 Computed Glycolysis Active Pathway of the *E. coli* pathway 100

List of Tables

4.1 Comparison with a CSP Solver [107] 65

4.2 Results for 2SPP Instances: `HT09`, `NGCUT09` 68

4.3 Comparison between Combined Methods 69

4.4 Comparison with the Exact Method by Alvarez-Valdes *et al.* [4] 72

4.5 Comparison with the Exact Method by Kenmochi*et al.* [83] 74

5.1 Size of Clauses of Ψ_z 89

5.2 Results for Pathways from the Literature [14] 96

5.3 Computed Minimal Active Pathways 99

Chapter 1

Introduction

1.1 Background

This thesis dedicates to a study of a solving method for optimization and enumeration problems by computing satisfiability and models of encoded propositional formulas.

Optimization is an important task in our life. In fact, we go through our days completing many optimization tasks, though we are not aware of it. For instance, we can see such tasks in a trip. There will be our demands, e.g., to arrive to a destination as early as possible, to see around tourist spots as much as possible in a limited time. In other words, the former demand is to minimize its traveling time and the later is to maximize the number of spots to see. These kinds of optimization tasks are also essential for industry; thus, it has been studied in many research domains.

Enumeration is another important task. We can sometimes not decide a parameter to be optimized. In this case, we may want to generate possible solutions to choose one from the other candidates. Enumeration is also useful when we want to have all solutions that give us utility information, such as statistics, common parts etc. Although both optimization and enumeration problems are important, some of them are strongly believed that there is no efficient procedure i.e. these problems cannot

be solved in polynomial time.

We thus need to use heuristics for solving these problems effectively. Moreover, considering practical applications, the following two conditions are necessary: being a general method that is applicable to several kinds of problems, being a flexible method that allows addition of constraints. As such a method to solve these problems, this thesis studies a method based on technologies that have been studied in a research domain of propositional satisfiability (SAT). In the following section, we provide an overview of SAT research.

1.1.1 Overview of SAT

The propositional satisfiability problem (or the Boolean satisfiability problem), the SAT problem for short, is a problem that has been paid much attention in computer science. When a propositional formula is given, the solution of the SAT problem is whether there is an assignment satisfying the given formula or not. Many researches from not only computational logic but also electronic design automation have studied SAT technologies until today. We briefly review its progress from mid-19th. Shannon first applies propositional logic to a design and simplification of electronic circuits in 1940 [123]. This was a novel work on the point that he used logic for representing electronic circuits because studies in logic were mainly motivated to explain human thought at that time.

Twenty years later, Davis and Putnam reported a paper containing several techniques each of which is inherited until today [35]. Specifically, they introduced three rules: (I) rule for the elimination of one-literal clauses, (II) affirmative-negative rule, and (III) rule for eliminating atomic formulas, each of which evolves into *unit clause rule*, *pure literal rule*, and *resolution* by other literature. Their procedure called DP consists of these rules is applied to conjunctive normal form formulas. Just two years later, an improved procedure called DPLL (Davis-Putnam-Logemann-Loveland) is re-

ported in the literature by Davis *et al.* [34]. Apparently, their procedure is similar to DP; the only difference is that they replace the rule (Ⅲ) with (Ⅲ*) *splitting rule*. It works with less memory compared with DP and becomes the basis of modern SAT solvers. In addition to these practical researches, the SAT problem is first established NP-complete problem by Cook [30]. This raises the importance of the SAT problem in computational complexity domain and it places in the center of computer science today.

Afterward, there have been developed many variations of DPLL and other type of SAT solvers. In particular, many solvers were proposed from 90's. Some researches focus on how to choose a decision variable. Jeroslow and Wang proposed a heuristic which roughly choose a literal appeared in a large number of short clauses in current formulas in 1990 [77]. Following this, improved heuristics in term of the performance of unsatisfiable problems are proposed by Freeman [49], and Böhm and Speckenmeyer [21]. In this age, in contrast to DPLL based solvers, Selman *et al.* proposed a SAT solver doing local search, which is called GSAT [121]. This solver becomes Walksat implementing an enhanced local search with random walk strategy [122]. Among studies for SAT, one important feature of SAT solvers is appeared in GRASP [94] that employs *clause learning*, which is an essential technique for modern SAT solvers.

Since 2000, the effective implementation of SAT algorithms becomes a popular topic as well as the search strategies. A SAT solver Chaff features two literal watching mechanism and it is outstanding to others [102]. Eén and Sörensson polished up implementations of previous solvers and provided an extensible search procedure that becomes the basic implementation of many solvers of nowadays [42]. In addition to those individual researches, SAT solver competitions are regularly held by year since 2002, which help to push up the quality of SAT solvers. These enormous researches trigger the extension of SAT researches as we describe in the next section.

1.1.2 SAT for Problem Solving

Since propositional formulas can directly represent electronic circuits as Shannon proposed, this is a major application of SAT solvers. For instance, the combinational equivalence checking, which is the problem of determining whether two given circuits implement the same function, is a typical one [57]. In these applications, SAT solver is mainly used for checking satisfiability of a given formula: the result of unsatisfiability tells us a given property holds; satisfiability tells us the existence of a counter example for the property.

Another remarkable aspect of SAT solvers is that almost all solvers can be used as a model generator for propositional formulas. That is, in addition to satisfiability, solvers can find a satisfiable assignment, a model, of a given propositional formula. It is very useful property for problem solving; it allows us to solve a variety of problems through propositional encoding. For instance, Crawford and Baker apply it to the job-shop scheduling problem [32]. They represent their problem in a propositional formula and obtain the problem solution by decoding a model of the formula. Since encoding methods affect problem solving performance, propositional encoding has gathered attention since '90s. For instance, in the research domain of constraint satisfaction problem (CSP), much effort has been spent for better encoding: there are many researches such as direct encoding [136], log encoding [76], support encoding [55], log support encoding [52], order encoding [127], and compact order encoding [129, 130]. In addition to above SAT-based approaches, there are several researches that utilize SAT search techniques. These researches do not encode their problem into propositional formulas but use recent techniques that are developed in SAT solving such as the DPLL procedure, clause learning, lazy data structure etc. For instance, the following researches include several SAT techniques: Answer Set Programming [92, 53], Quantified Boolean Formula [118] and MaxSAT [90, 66]. Nevertheless many SAT-based approaches have been studied, there are still challenging problems for which

not enough focus is paid. In particular, by multiple execution of a SAT solver, we can reach other types of problems. This topic is very attractive. In the next section, we explain this topic.

1.1.3 Incremental SAT Solving

Since the SAT problem is a decision problem, the SAT solver basically returns the satisfiability of a given formula. We thus need some extension to apply it to a wide range of problems. An extension is to incrementally execute a SAT solver to solve successive problems one by one, which we call incremental SAT solving. In 1992, Kautz and Selman first apply a SAT solver to solve a classical planning problem [81]. They divided the original problem into sub-problems so that each of them is a problem of deciding whether there is a plan of a given length or not. Then, they encode each sub-problem into a SAT problem. In their method, sub-problems are incrementally solved from a small length and the length of the problem that is firstly decided as satisfiable is the shortest plan length. In 1999, Biere *et al.* apply SAT techniques to symbolic model checking [19]. They focus the detection of counter examples of a fixed length and generate a SAT problem that is satisfiable when there is a counter example. They introduce the notion of bounded model checking; the bound corresponds to the maximum length of a counterexample and they increment the bound one by one. Following that, Eén and Sörensson apply their incremental method to temporal induction that checks safety properties on finite state machines. Crawford and Baker proposed an encoding method for the job-shop scheduling problem [32]. However, they treat it as a decision problem and do not mention a way to approach the optimal value. Soh *et al.* studies their encoding method and apply Multisat, which is a meta-heuristic solver, to the job-shop scheduling problem and show how it computes the optimal value of the problem [125, 74]. Besides above researches, Hooker proposed a variant of DPLL procedure that is adapted to incrementally solve multiple problems [70]. Bennaceur

et al. proposed an incremental branch-and-bound method which includes Lagrangean relaxations, meta-heuristics, and judicious jumping back [18].

Considering these approaches, we think that it has great potential and more studies are needed for the incremental SAT solving methods. In particular, we are interested in its potential for a variety of problems because of the recent progress of SAT technologies. Optimization problems are part of our target. Besides that, we apply it to a new domain as we describe in the next section.

1.1.4 Further Applications

Many researchers from Computer Science start to apply their own technology to biology. In addition to well known combinatorial problems, we think that SAT-based approach, particularly incremental SAT solving, can be an effective approach for problems of biology. We here particularly focus on a novel research domain called *Systems Biology*, which is expected to be a key approach to promote a more systematic understanding of living cell and life.

An accepted way in Systems Biology is to capture the interactions of biological objects by a network. It is a strong tool and has been studied by a lot of researches; there are various ways to interpret biological interactions to it. A longstanding approach is to represent such networks as a system of differential equations. These equations are constructed from a given network using laws in chemistry and biology, such as laws of mass action, Michaelis-Menten kinetics. This method allows detailed analyses e.g. concentrations of each metabolite with time variation on continuous values. However, there are some problems. At the first place, it is difficult to measure precise value in organisms. Besides, it is sometimes difficult to develop differential equations capturing interactions of biological objects due to its difficult parameter tuning. Moreover, although there are attempts, it is not easy task to compute a solution of differential equations and not scalable consequently.

On the other hand, systems of networks over discrete values, rather than continuous values, have been proposed. In the following, we review those methods. As such a method, Petri net has been used to represent systems of networks. It is originated by Carl Adam Petri in his thesis in 1962 [108], and used in a wide range of research areas today. General aspect of the Petri net is well summarized by Desel and Juhas [39]. It is first applied to systems biology in the literature in 1993 [117]. Following this, Hofestädt report another one [69]. Recent progress on systems biology using Petri net is summarized in the literature [98, 27]. In addition to Petri net approaches, there are some researches on analyses of metabolic pathways. In 2000, the elementary flux mode analysis is proposed and has been widely used for analyses of metabolic pathways until today [120]. This method can treat multi-molecular reactions while taking into account stoichiometry. Beasley and Planes proposed an optimization approach to metabolic pathway analyses [14]. They perceived stoichiometry of reactions by constraints and modeled possible pathways as optimized solutions.

Although researches have been proposed in Systems Biology, there are still some problems. One of them is scalability; it is still difficult to analyze cell scale pathways in many methods. Although elementary flux mode analyses can treat large pathways, it generates a number of solutions that is not tractable to analyze. To analyze cell scale pathways, we think that a method equips following features is necessary: scalable to a cell scale pathway; capable to treat biological constraints; flexible to allow additional constraints. We believe that our method based on SAT technologies is a key method that qualifies those features.

1.2 Contributions

In this thesis, we study a general framework of incremental SAT solving: it incrementally computes the satisfiability and models of propositional formulas until a given

goal condition is satisfied. We then review how previous approaches utilizing SAT technologies are represented in this framework. The main two contributions of this thesis are applying this incremental SAT solving to the following two problems.

First contribution is our study on applying incremental SAT solving to solve the two-dimensional strip packing problem (2SPP). In this problem, we are given a set of rectangles and one large rectangle called a strip. The goal of the problem is to pack all rectangles without overlapping, into the strip by minimizing the overall height of the packing. Although the 2SPP has been studied in Operations Research, some instances are still hard to solve. Our method solves the 2SPP by translating it into SAT problems through a propositional encoding called order encoding. The translated SAT problems tend to be large; thus, we apply several techniques to reduce the search space by symmetry breaking and positional relations of rectangles. To solve the 2SPP, that is, to compute the minimum height of a 2SPP, we need to repeatedly solve similar SAT problems. We thus apply incremental SAT solving to 2SPP. To evaluate our approach, we obtain results for 38 instances from the literature and made comparisons with a state-of-the-art constraint satisfaction solver and an ad-hoc 2SPP solver.

Second contribution is our study on applying incremental SAT solving to identify necessary reaction sets in metabolic pathways by minimal model generation. In systems biology, identifying vital functions like glycolysis from a given metabolic pathway is important for better understanding of living organisms. We particularly focus on the problem of finding minimal active pathways producing target metabolites from source metabolites. We translate laws of biochemical reactions into propositional formulas and apply incremental SAT solving to compute its minimal models. An advantage of our method is that it can treat reversible reactions and cycles. Moreover, the translation enables us to obtain solutions for large pathways. We apply our method to a whole *Escherichia coli* metabolic pathway. As a result, we have found the conventional glycolysis pathway described in a biological database EcoCyc.

1.3 Thesis Organization

The organization of this thesis is as follows. Section 2 provides an introduction of propositional logic and SAT problems. We then describe SAT technologies; in particular, we focus on the mechanism of conflict-driven clause learning (CDCL) solver based on the DPLL procedure. This type of solver is used through the remainder of this thesis. Following that, we explain model generation based on SAT solvers. The details of this procedure are also given in this section. Although recent progress of SAT solvers is enormous, we need an extension to apply it to a wide range of problems. To do that, we introduce incremental SAT solving. Section 3 provides how to apply incremental SAT solving to optimization and enumeration problems. We also give a better way to solve this problem by reusing learned clauses. In Section 4, we apply incremental SAT solving to solve the two-dimensional strip packing problems. For this problem, a SAT encoding method called order encoding is introduced. We briefly give the explanation of this encoding and discuss why this is effective compared with other encoding methods. In Section 5, as a new application of SAT technologies, the minimal active pathway finding problem is introduced. We then apply incremental SAT solving to this problem. In experiments, we give not only a computational evaluation but also a biological evaluation. Finally, Section 6 gives the conclusion of this thesis and a discussion of future research topics.

1.4 Publications

The referred publications are as follows:

1. Takehide Soh and Katsumi Inoue. Identifying Necessary Reactions in Metabolic Pathways by Minimal Model Generation. *The Sixth Conference on Prestigious Applications of Intelligent Systems*, In *the Proceedings of European Conference of Artificial Intelligence 2010*, pp.277–282, Lisbon, 2010.

2. Takehide Soh, Katsumi Inoue, Naoyuki Tamura, Mutsunori Banbara, and Hidetomo Nabeshima. A SAT-based Method for Solving the Two-dimensional Strip Packing Problem. *Fundamenta Informaticae*, 102(3–4): 467–487, IOS Press, 2010.

3. Takehide Soh and Katsumi Inoue. Finding Minimal Reaction Sets in Large Metabolic Pathways. In *the Proceedings of the Workshop on Constraint Based Methods for Bioinformatics (WCB 2010)*, pp.54-68, Edinburgh, 2010.

4. Takehide Soh and Katsumi Inoue. A SAT-based Method for Analyzing Metabolic Pathways. *Poster presentation at: Systems Biochemistry 2010*, University of York, 2010.

5. Hidetomo Nabeshima and Takehide Soh. Principles of Modern SAT Solvers (in Japanese). Tutorial Paper, *Journal of the Japan Society for Artificial Intelligence*, 25(1):68-76, 2010.

6. Takehide Soh, Katsumi Inoue, Naoyuki Tamura, Mutsunori Banbara, Hidetomo Nabeshima. A SAT-based Method for Solving the Two-dimensional Strip Packing Problem. In Marco Gavanelli and Toni Mancini (eds.), *the Proceedings of 15th RCRA workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion*, No.16 (15 pages), CEUR Workshop Proceedings (CEUR-WS.org), Vol.451, 2008.

7. Hidetomo Nabeshima, Takehide Soh, Katsumi Inoue, and Koji Iwanuma. Lemma Reusing for SAT based Planning and Scheduling. In *the proceedings of the International Conference on Automated Planning and Scheduling 2006 (ICAPS 2006)*, pp. 103–112, 2006.

8. Katsumi Inoue, Takehide Soh, Seiji Ueda, Yoshito Sasaura, Mutsunori Banbara and Naoyuki Tamura. A Competitive and Cooperative Approach to Propositional Satisfiability. *Discrete Applied Mathematics*, 154(16):2291–2306, Elsevier, 2006.

9. Takehide Soh, Katsumi Inoue, Mutsunori Banbara, and Naoyuki Tamura. Experimental Results for Solving Job-shop Scheduling Problems with Multiple SAT Solvers. In *the Proceedings of the 1st International Workshop on Distributed and Speculative Constraint Processing (held in conjunction with CP 2005)*, pp. 25–38, Sitges Spain, 2005.

Chapter 2

Propositional Logic and SAT

In this chapter, we give an explanation of propositional logic. Following that, the definition of the SAT problem is given. We then introduce foundations of SAT solvers; in particular, we focus on the mechanism of conflict-driven clause learning (CDCL) solvers. Above contents in Section 2.1 and 2.2 are adapted from [20, 26, 75, 103, 134]. Almost all SAT solvers can be seen as a model generator. We briefly explain this feature and give how to generate minimal models with SAT solvers in the end of this chapter.

2.1 Propositional Formulas and its Satisfiability

2.1.1 Syntax

A *proposition* is a sentence that can be evaluated as either true or false. A *propositional variable* is a symbolic representation of a proposition. We often represent propositional variables in alphabetical symbols such as, p, q, $r, \ldots, x, y$. We represent the set of propositional variables in V. Logical connectives consist of the unary connective $\neg$ and binary connectives $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$. A *propositional formula* consists of propositional variables and logical connectives.

Definition 1 Propositional formulas are defined as follows:

1. A propositional variable $p \in V$ is a propositional formula.
2. If α and β are propositional formulas, then: $\neg\alpha$, $\alpha \vee \beta$, $\alpha \wedge \beta$, $\alpha \rightarrow \beta$ and $\alpha \leftrightarrow \beta$ are propositional formulas.

Above definition gives a general form of propositional formulas. We then introduce a special form of formulas called conjunctive normal form (CNF), which is widely used in computational logic. We give necessary definitions as follows. A *literal* is either a variable or its negation. A literal is *positive* if it is a propositional variable, and is *negative* if it is a negation of a propositional variable. For a literal l, $\bar{l}$ denotes the *complement* of l, i.e., when p is a propositional variable, $\overline{p} = \neg p$ and $\overline{\neg p} = p$. A *clause* is a disjunction of literals. The *length* of a clause is the number of literals in the clause. The clause of length zero is called the *empty clause*. In the rest of this thesis, we sometimes use the symbol $\Box$ as the empty clause. A *unit clause* is a clause of length one. A literal *appears positively* (or *negatively*) in a clause if it appears as a positive (or negative) literal in the clause. A *conjunctive normal form (CNF) formula* is a conjunction of clauses. A CNF formula is identified with a set of clauses, and is also simply called a *formula* in the rest of this thesis.

In SAT research domain, this CNF is usually used. Any propositional formula can be converted into a CNF formula in exponential time. We thus can limit the form to CNF without loss of generality. However, sometimes the number of converted clauses becomes huge. Tseitin thus provides a translation method from any propositional formula into an equi-satisfiable CNF formula [111]. In the method, we introduce new variables that represent sub-formula of the given original formula. For instance, Suppose that $F = (p \wedge q) \vee (r \wedge s)$ is given. We then introduce two new propositional variables x and y and clauses representing $x \leftrightarrow (p \wedge q)$ and $y \leftrightarrow (r \wedge s)$. Finally, the translated formula will be $F' = (x \vee y) \wedge (\neg x \vee p) \wedge (\neg x \vee q) \wedge (\neg p \vee \neg q \vee x) \wedge (\neg y \vee r) \wedge (\neg y \vee s) \wedge (\neg r \vee \neg s \vee y)$.

2.1.2 Semantics

The symbols $True$ and $False$ are called *truth values*. We often use abbreviations for truth values such as T for $True$ and F for $False$. An *interpretation* (or an *assignment*) of a propositional formula is given by a (partial) mapping $I : V \to \{T, F\}$.

Definition 2 Let ψ, α, β be propositional formulas over V. Let I be an interpretation. We say that I *satisfies* ψ, and write $I \models \psi$, if $True$ is assigned to ψ by I. We also write $I \not\models \psi$, otherwise. The truth value of ψ is recursively defined as follows:

1. $I \models p$ if and only if $I(p) =$T, where $p \in V$.
2. $I \models \neg\psi$ if and only if $I \not\models \psi$.
3. $I \models \alpha \vee \beta$ if and only if $I \models \alpha$ or $I \models \beta$.
4. $I \models \alpha \wedge \beta$ if and only if $I \models \alpha$ and $I \models \beta$.

Definition 3 When an interpretation I satisfies a propositional formula ψ, we say that I is a *model* of ψ. In this case, we also say that ψ is *true* in I.

Definition 4 Let Σ be a set of propositional formulas, and I satisfies for every formula $\psi \in \Sigma$. Then I is a model of Σ. If Σ has a model then Σ is *satisfiable*. Otherwise, it is *unsatisfiable*.

Above definitions give semantics for general propositional formulas. Accordingly, we can say followings for CNF formulas: For a negative literal $\neg p$, $I \models \neg p$ iff $I(p) = F$. For a clause $c = l_1 \vee \cdots \vee l_n$, $I \models c$ iff $I \models l_i$ for some literal l_i $(1 \leq i \leq n)$. For a CNF formula $\psi = c_1 \wedge \cdots \wedge c_m$, $I \models \psi$ iff $I \models c_j$ for every clause c_j $(1 \leq j \leq m)$.

Following semantics of propositional formulas, we give the definition of the SAT problem in the next section.

2.1.3 SAT Problem

The propositional (or Boolean) satisfiability (SAT) is well known as the first established NP-complete problem [30]. Each SAT instance is a propositional CNF formula, and is called a *SAT problem*. The definition is given as follows.

Definition 5 *Propositional satisfiability*

Input. A propositional formula ψ.

Output. The satisfiability of ψ.

For SAT problems, we use abbreviation `SAT` and `UNSAT` to represent "satisfiable" and "unsatisfiable," respectively. For instance, suppose that the following formula ψ is the input of a SAT problem:

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (x_3 \vee x_4) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_4).$$

A satisfiable assignment $(x_1, T), (x_2, T), (x_3, T), (x_4, F)$ exists for ψ; thus, the output of the problem is satisfiable (`SAT`).

2.2 Modern SAT Solvers and Techniques

In this section, we review the progress of SAT techniques. As said before, we use a word *SAT* as the decision problem of satisfiability for propositional formulas in conjunctive normal form (CNF). Also, we call each SAT instance a *SAT problem*, and call a procedure for SAT a *SAT solver*.

Many researches on SAT solvers have been made particularly since 90's. Thanks to such progress, the state-of-the-art solver can solve problems that consist of over ten million clauses. Surprisingly, although a number of SAT solvers have been developed, almost all SAT solvers originated from a procedure called DPLL (Davis-Putnam-Logemann-Loveland)[34], which is an algorithm with *backtracking* proposed in 1962.

Since then, many improvements have been made to the procedure. In the beginning, some researches focus on how to choose a decision variable. For instance, Jeroslow and Wang proposed a heuristic which roughly choose a literal appeared in a large number of short clauses in current formulas [77]. In addition, several solvers with look-ahead variable selection are proposed [49, 21, 91, 38, 67]. In the late 90's, a *clause learning* mechanism is first employed to a SAT solver GRASP, which analyzes a conflict of the current truth assignment for literals [94]. Following that, the effective implementation of SAT algorithms becomes a popular topic as well as the search strategies. Chaff features two literal watching scheme and it outstands others [102]. Eén and Sörensson polished it up and provided an extensible search procedure which becomes the basis of many solvers of nowadays [42]. Those solvers described in the above, performs *systematic search*, and can decide both the satisfiability and the unsatisfiability of SAT problems. When a SAT problem is unsatisfiable, a complete solver terminates once all possible truth assignments have been examined, proving that the problem is unsatisfiable.

In the following section, we review conflict-driven clause learning (CDCL) solvers following the DPLL procedure. We particularly focus on the clause learning mechanism, which is employed in incremental SAT solving described in the next chapter.

2.2.1 DPLL Procedure

Before the explanation of DPLL, we give preliminaries. A propositional formula ψ is given as a CNF formula over the set of variables V. Let x be a propositional variable such that $x \in V$. Let σ be an assignment to the set V such that: $\sigma : V \rightarrow \{T, F\}$. Let ρ be a partial assignment for a formula ψ. Those assignments are also represented in a set of pairs of variables and assignments such that (x, T). We also introduce two mappings related to literals. Let var a mapping such that $var(x) = x$ and $var(\neg x) = x$. Let $sign$ a mapping such that $sign(x) = T$ and $sign(\neg x) = F$.

```
DPLL(CNF ψ, Partial Assignment ρ)
    begin
1:      (ψ, ρ) := UnitPropagation(ψ, ρ);
2:      if ψ is empty then
3:          output ρ;
4:          return SAT;
5:      if □ ∈ ψ then return UNSAT;
6:      Select an unassigned variable x by some heuristic;
7:      if DPLL(ψ, ρ ∪ {(x, T)}) is SAT then return SAT;
8:      return DPLL(ψ, ρ ∪ {(x, F)});
    end

UnitPropagation(CNF ψ, Partial Assignment ρ )
    begin
9:      while □ ∉ ψ and some unit clause {l} exists in ψ do
10:         ρ := ρ ∪ {(var(l), sign(l))};
11:         ψ := ψ|ρ;
12:     return (ψ, ρ);
    end
```

Figure 2.1: DPLL Algorithm

We denote the simplified formula obtained by a partial assignment ρ as $\psi|_\rho$. In the simplified formula $\psi|_\rho$, satisfied clauses by ρ are removed, and in the remaining clauses, literals assigned to F by ρ are removed from ψ. Suppose that a CNF formula $\psi = (x_1 \vee x_3 \vee \neg x_4) \wedge (x_4) \wedge (x_2 \vee \neg x_3)$ is given and a partial truth assignment ρ is a set $\{(x_4, T)\}$. Then $\psi|_\rho = (x_1 \vee x_3) \wedge (x_2 \vee \neg x_3)$.

The DPLL algorithm is shown in Figure 2.1. The algorithm DPLL takes a CNF-formula ψ and a partial assignment as inputs. We explain this algorithm as follows.

At Line 1, a procedure UnitPropagation(ψ, ρ) is invoked. The procedure simplifies the input ψ with every unit clause $\{l\} \in \psi$ (Line 9-12). Specifically, the partial assignment ρ is extended to $\rho \cup \{(var(l), sign(l))\}$ by an unit clause $\{l\}$ (Line 10). Then, ψ is simplified to $\psi|_\rho$ (Line 11). In this simplification, a *conflict* is detected whenever empty clause is detected. These assignment extension and simplification are iterated until (i) no more propagation is possible or (ii) an empty clause is detected. The end of the unit propagation, the simplified formula ψ and the extended assignment ρ are returned to the DPLL procedure.

At Line 2, the result of UnitPropagation is checked: if $\psi|_\rho$ is empty, the assignment ρ is outputted and `SAT` is returned. At Line 5, if ψ contains the empty clause then `UNSAT` is returned. Otherwise, at Line 6, the DPLL selects a variable x by some heuristic, such as the one by Jeroslow and Wang [77], and MOMS [49]. In the next, if DPLL$(\psi, \rho \cup \{(x, T)\})$ returns `SAT`, then `SAT` is returned (Line 7); otherwise, return the result of DPLL$(\psi, \rho \cup \{(x, F)\})$ at Line 8. Accordingly, the procedure returns `SAT` whenever ψ is empty, or it returners `UNSAT` if it cannot find satisfiable assignment after an exhaustive search by recursively invoking DPLL. Nowadays, almost all modern SAT solvers are based on this procedure and Many improvements have been added since it was born. In particular, a clause learning mechanism extends its ability to be applicable to large SAT problems. In the next section, we explain those solvers employ this mechanism.

2.2.2 CDCL Solvers and Techniques

In recent years, almost all DPLL-based SAT solvers implement a clause learning mechanism. Since this mechanism controls its entire procedure, we call such solvers conflict-driven clause learning (CDCL) solvers. Before the explanation of the algorithm of CDCL Solvers, we give some preliminaries. As we described in the previous section, DPLL-based procedures including CDCL Solvers iterate UnitPropagation and the vari-

able selection. During such a process, the value of each variable is assigned either by two ways. We distinguish those variables as follows: (I) a variable assigned by a variable selection process is called a *decision variable*; (II) a variable assigned by UnitPropagation is called an *implied variable*. As same as the previous section, we use ρ to represent current partial assignments. We call the number of decision variables included in ρ the current *decision level*. Let *level* be a mapping such that $level(x)$ denotes the decision level where the value of the variable x is assigned.

The algorithm of the CDCL Solver is given in Figure 2.2. At Line 1, the decision level *Dlevel* and the assignment ρ is initialized. Then the loop begin and the first UnitPropagation runs and returns the simplified formula ψ and the updated assignment ρ. The process then goes either of three ways according to the state of ψ (Line 4, 7 or 13): (a) the formula is satisfied; (b) the empty clause is detected i.e. a conflict occurs; (c)otherwise. In the following, we explain each branch (a), (b) and (c). (a) If the formula is empty then it outputs ρ and returns `SAT` (Line 5-6). This operation is same as the DPLL procedure. The difference appears in how to treat the obtained conflict. (b) If a conflict is detected at Line 7, then the current decision level is checked (Line 8). If the level is equal to 0 that means there is no assignment to satisfy the given formula, then `UNSAT` is returned. Otherwise, a procedure ConflictAnalysis is invoked to analyze the conflict. The procedure decide the backtrack level and generate a learned clause that permanently avoids the same conflict (Line 9). Following that, the formula ψ, the current assignment ρ, and the decision level are updated (Line 10-13). We give a detailed explanation of the procedure ConflictAnalysis in the next section. (c) Otherwise, the situation must be that there is no empty clause and there are remaining unsatisfied clauses. Then, the decision level is counted up and the procedure updates the assignment ρ with an unassigned variable x and its value *value* selected by some heuristic, which is typically the one called VSIDS first proposed in the literature [102] (Line 15-17). Above operations are iterated until ψ becomes

```
CDCL Solver(CNF ψ)
    begin
1:      Dlevel := 0; ρ := {};
2:      while (TRUE) do
3:          (ψ, ρ) := UnitPropagation(ψ, ρ);
4:          if ψ is empty then
5:              output ρ;
6:              return SAT;
7:          else if □ ∈ ψ then
8:              if Dlevel = 0 then return UNSAT;
9:              (c, Blevel) := ConflictAnalysis(ψ, ρ);
10:             Removed literals and clauses after Blebel in ψ are recovered;
11:             ψ := ψ ∪ {c}
12:             ρ := {(x, value) ∈ ρ | level(x) ≤ Blevel};
13:             Dlevel := Blebel;
14:         else
15:             Dlevel := Dlevel + 1;
16:             select an unassigned variable x and its value by some heuristic;
17:             ρ := ρ ∪ {(x, value)};
    end
```

Figure 2.2: Algorithm of CDCL Solvers

empty or a conflict is detected at the decision level zero. An important feature of the algorithm is ConflictAnalysis. It dramatically enhances the performance of SAT solvers. In the following section, we explain how it works.

2.2.3 Clause Learning

Since the appearance of GRASP and Relsat [94, 11], clause learning is one of the most essential techniques to accelerate SAT solving. In this section, we explain how CDCL solvers generate learned clauses. In CDCL solvers, an *implication graph* is processed implicitly during UnitPropagation. Let N be a set of nodes representing assignments and decision levels for every assigned variable. For instance, a node representing a variable x that is assigned to F at the decision level i is labeled as $\neg x@i$. Let A a set of arcs representing the propagation occurred in the current UnitPropagation. Let G be a directed graph such that $G = (N, A)$. Let $Dlevel$ be the current decision level. The implication graph G is constructed as follows:

1. N is initialized as a set of nodes representing assignments to variables whose decision levels lv such that $lv < Dlevel$.

2. add a node representing the assignment of the latest decision variable to N.

3. While there exists a clause $C = (l_1 \vee \cdots \vee l_k \vee l))$ satisfying that nodes of $\overline{l_1}, \ldots, \overline{l_k}$ have already existed in N:

   (a) add a node labeled $l@Dlevel$ to N if it does not exist in N.

   (b) add arcs $\{(l_i@j, l@Dlevel) | 1 \leq i \leq k, j \leq Dlevel, l_i@j \in N, \}$, if each of them does not exists in A.

   (c) if two nodes $l@Dlevel$ and $\bar{l}@Dlevel$ simultaneously exist in N, i.e., a conflict is detected, then the construction of G is terminated and $var(l)$ is generated as a *conflict variable*.

If a conflict of x is detected during the construction of G, then it is called a *conflict graph* and we call x and $\neg x$ *conflict literals*. In CDCL solvers, an implication graph (or a conflict graph) is constructed during the UnitPropagation. If a conflict is detected, we then analyze the conflict graph and obtain a *learned clause* as follows. At first,

the graph is split into two sub-graphs: *reason side* and *conflict side*. The reason side includes all decision variables; the conflict side includes the conflict literals. Usually, there are many ways to split the conflict graph. We here explain a de-fact standard. Suppose that paths from the latest decision variable to the conflict variable. The *unique implication points* (UIP) are the nodes which all such paths include. First UIP is the UIP which is the closest to the conflict variable. In the literature [141], Zhang *et al.* empirically proved that it is better to split the graph just after the first UIP. Nowadays, first UIP is the accepted scheme for SAT solvers. The leaned clause is constructed from the complement of each literal corresponding to nodes on the reason side that have at least one edge going to the conflict side. In the following we explain the learned clause generation with a specific example.

Example of Learned Clause Generation

We here explain a generation of learned clauses with the following 9 clauses.

$$c_1 = \{\neg x_1, \neg x_5, x_8, x_{11}\} \qquad c_6 = \{x_7, \neg x_8\}$$
$$c_2 = \{\neg x_2, \neg x_3, \neg x_6\} \qquad c_7 = \{x_5, x_7\}$$
$$c_3 = \{\neg x_2, x_4, x_6, x_{10}\} \qquad c_8 = \{x_7, \neg x_9\}$$
$$c_4 = \{\neg x_3, \neg x_4\} \qquad c_9 = \{x_9, \neg x_{11}\}$$
$$c_5 = \{\neg x_7, \neg x_{10}\}$$

Suppose that x_1, x_2 and x_3 are selected as decision variables, and each of them is assigned to T. When x_3 is assigned to T, then unit propagation makes x_4 to F by c_4 and x_6 to F by c_2, and this propagation cause a conflict of the assignment of x_{11}. This sequence of assignments is shown as a conflict graph in Figure 2.3. We repeat that each node represents each assignment to a variable with its decision level. Black nodes represent variables assigned at earlier decision levels and white nodes represent variables assigned in the current level. In this case, the conflict occurs at the decision

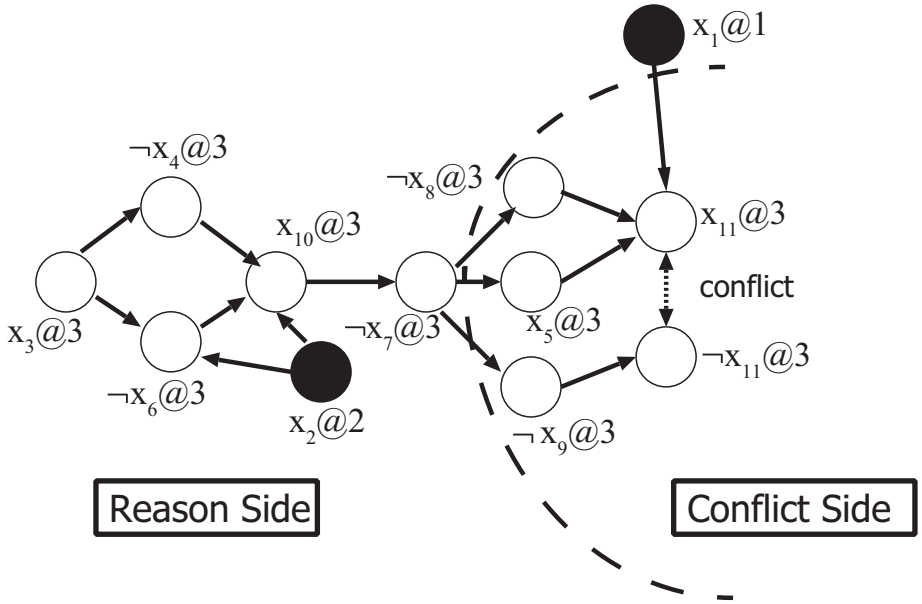

Figure 2.3: Example of an Implication Graph

level 3. Then we need to split the implication graph into two sides: (i) one is the conflict side which includes the conflict variable and does not include black nodes, (ii) the other one is the reason side. In Figure 2.3, x_7 is the first UIP. The learned clause is generated based on the set of literals which have arcs from reason side to conflict side, that is, $\{x_1, \neg x_7\}$ which represents the reason of the conflict. Finally, $c = \{\neg x_1, x_7\}$ is generated as a learned clause. According to the decision levels of the literals included in the learned clause, the value of x_7 can be decided in the decision level one. In this case, the procedure ConflictAnalysis returns one as the backtrack level and c as the learned clause.

2.2.4 Other Important Techniques

Watched Literals

The procedure of UnitPropagation dominates large portion of the computational time of SAT solvers. To accelerate UnitPropagation is a critical mission for all complete

SAT solvers. The first thing to do in the procedure is to detect unit clauses. A naive method is as follows. For each variables x, the method prepare a list that represents a set of clauses including the variable x. When some value is assigned to x, the method searches the list of x and detects the following status: (a) an unit clause, (b) satisfied, (c) conflicted, and (d) others. This method is implemented in GRASP [94] and Relsat [11]. In practice, we want to find only the status of (a) and (c); however, almost all clauses are neither (a) nor (b). To resolve this problem, another method called *watched literals* is introduced in Chaff [102]. Let c be a clause $c = \{l_1, \ldots, l_n\}$. A clause c is called *active* unless the clause is removed from the original formula, that is, the clause is satisfied under the current assignment. The watched literal scheme tries to keep two literals l_i, l_j which are not yet assigned in each active clause. Note that in the CDCL procedure, detected unit clauses and satisfied clauses are immediately removed from a given formula; thus, the scheme can always have such two literals in all active clauses. When one of the literals l_i is assigned to T, all clauses including l_i is satisfied and removed from the formula; when one of the literals l_i is assigned to F, the scheme performs either two operations: (i) if there is some unassigned literal l_k then the watched two literals will be changed to l_k and l_j, (ii) if there is no unassigned literal then the clause is detected as an unit clause. By this scheme, we only need to search watched literals in active clauses to detect unit clauses and conflicts.

The benefits of this scheme are summarized as follows. First, when some value is assigned to a variable x, we only need to search clauses watching x. It extremely reduces the number of clauses compared with the naive method. Second, we need to perform nothing when the procedure backtracks. This watched literal scheme becomes an essential function of modern SAT solvers as well as clause learning.

Restart Strategy

Since the SAT problem is NP-complete, SAT solvers need to implement several heuristics and the behavior of solvers is difficult to expect. For instance, some problem can easily be solved but it can sometimes be difficult even if we only replace names of variables. This phenomenon that the behavior of SAT solvers surprisingly changed with a little possibility is known as heavy-tailed behavior [59]. To make SAT solvers behavior stable, Gomes *et al.* suggested it is effective to restart a process by a relatively short span [60]. In CDCL solvers, the span is often decided by the number of conflicts. At the restart, it backtracks to the decision level zero and restart search with conflict clauses obtained so far. By keeping conflict clauses, it can avoid previous fails. Besides, to ensure the completeness of the procedure, the span is getting longer within the process. Since the appearance of Chaff, many SAT solvers employ frequent restart strategies [109, 5]. A comparison of restart strategies are well summarized in [72].

Variable Selection

When a CDCL solver cannot decide the satisfiability of a given problem after the first UnitPropagation, it needs to pick a variable and assign some value to it. How do we choose such variable? It is very difficult problem and it extremely affects the performance of SAT solvers. There have been several heuristics are proposed, such as maximum occurrence in clauses of minimum size (MOMS) [49], BOHM [21], and variable state independent decaying sum (VSIDS) [102].

In particular, we here describe the heuristics called VSIDS [102], which is commonly used in most of CDCL solver as follows: (a) the heuristic uses counters for each literals l in a given formula and gives zero to it, (b) each counter is increased by every generation of a learned clause including the literal l, (c) by every fixed span, the value of every counter is divided by a constant. At the variable selection in a solver process,

the heuristic performs either two operations: (i) pick up the literal with the highest counter, or (ii) randomly pick up a literal with some probability. By (b) and (c), the heuristic tends to choose a variable included in learned clauses that are recently generated by conflicts. It accelerates local search around the conflict; consequently, it helps to resolve the conflict. Since values of counters are not changed unless a conflict is occurred, the VSIDS heuristic focuses to resolve a conflict compared with MOMS and BOHM. Other feature of VSIDS is its less small computational cost because we only need to increase each counter when a conflict is occurred.

2.3 Model Generation by SAT Solvers

So far, we briefly explain CDCL solvers. In this section, we first explain several model generators in propositional logic. We then explain that SAT solvers are also used as a propositional model generator. Besides model generation, there is a research topic of model counting (or #SAT) based on DPLL, which is the problem of computing only the number of models. It is also studied actively and applications to probabilistic inference are reported in recent years [6, 139, 56] but we do not treat this topic here.

In 1986, Bryant proposed a Boolean function representation called the reduced ordered binary decision diagrams (ROBDDs), which recently expands to several research domains in computer science [23]. It is also utilized as a model generator applied to model checking [24]. Following that, there are some attempts utilizing the techniques of BDDs to solve SAT problems as well as model generation [114, 61]. Besides, Hasegawa *et al.* propose a model generation procedure called MGTP [64, 63]. In particular, MGTP computes minimal models, which are explained later. To compute such models, it starts with an empty set as an initial model candidate. If the candidate does not satisfy a given model, MGTP then extends the candidate. This procedure is repeated until a model is found or unsatisfiability is confirmed.

Model Generation (Ψ)

begin

1: $\Sigma := \emptyset$;

2: result := `SAT`

3: **while** (result != `UNSAT`)

4: (result, I) = $\mathsf{Solve}(\Psi)$;

5: **if** result = `SAT` **then**

6: $\Sigma := \Sigma \cup \{I\}$;

7: $\psi_{block} := \bigvee_{p \in I} \neg p \vee \bigvee_{q \in \overline{I}} q$;

8: $\Psi := \Psi \wedge \psi_{block}$;

9: return Σ;

end

Figure 2.4: Model Generation

In addition to approaches described above, since a model of a propositional formula is given as its satisfiable assignment, almost all SAT solvers can be seen as a model generator. Besides, there are some researches for all model enumeration by SAT solvers [78, 101, 86, 95].

In the following, we explain model generation methods using SAT solvers. Let $V = \{x_1, x_2, \ldots, x_i\}$ be a set of propositional variables that are in a formula Ψ. As we explained in the previous section, a model for a CNF formula Ψ is an assignment σ where all clauses are satisfied. In addition to this mapping representation, models can also be represented in the set of propositional variables to which it maps $True$. Suppose that I is a model and $\overline{I}$ is its complement such that $\overline{I} = V \setminus I$. Then, the model mapping x_1 to T, x_2 to F, x_3 to T is represented by the set $I = \{x_1, x_3\}$; its complement is represented by the set $\overline{I} = \{x_2\}$. In the remainder of this chapter, we

represent a model in this set notation.

Then, the model generation (enumeration) algorithm is then shown in Figure 2.4. In the algorithm, Σ denotes a set of models. Line 1 and 2 are for initialization. Then, the first model is obtained at Line 4 if a given formula Ψ is satisfiable, and the model is added to Σ (Line 6). Following that, a *block clause* is constructed at Line 7 to avoid to generate same models. Although we here show the simplest block clause, refinements have been studied in [78, 101]. Following that, the block clause is added to Ψ at Line 8. This procedure is iterated until `UNSAT` is returned by solve. Finally, the set of models of Ψ is returned as Σ.

In some applications, we sometimes want more essential models in which we are really interested instead of computing all models. For this purpose, a notion of *minimal* is useful. We here show how to generate *minimal models* by a SAT solver. We start the definition of minimal models and its preliminaries.

Definition 6 Let $V_p \subset V$ be a set of propositional variables and Ψ a CNF formula. A model I is a *minimal model* of Ψ with respect to V_p iff I is a model of Ψ and there is no model I' of Ψ such that $I' \cap V_p \subset I \cap V_p$.

For instance, suppose that Ψ is a propositional formula $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3)$. Then, all models of Ψ are $\{x_1\}$, $\{x_2, x_3\}$, $\{x_1, x_3\}$ and the minimal models of Ψ are $\{x_1\}$ and $\{x_2, x_3\}$.

Intuitively, minimal models are mutually excluded and these are more essential than other models. Niemelä report a theorem which is the basis of the computational treatment of *minimal models* [106]. Koshimura *et al.* extends one of his propositions [106], which gives a method to compute a minimal model with respect to a set of propositional variables [86]. The theorem [86] is given as follows:

Theorem 1 Let Ψ be a CNF formula, I a model of Ψ, and V_p a set of propositional variables. I is a minimal model of ψ with respect to V_p iff a formula $\Psi_c = \Psi \wedge \neg(x_1 \wedge$

Minimal Model Generation (Ψ, V_p)

begin

1: $\Sigma := \emptyset$;
2: **while** (True)
3: (result, I) := Solve(Ψ) ;
4: **if** result = `UNSAT` **then** return Σ ;
5: **else**
6: $V_x := I \cap V_p$;
7: $V_y := \overline{I} \cap V_p$;
8: $\Psi_c := \Psi \wedge \left(\bigvee_{x_i \in V_x} \neg x_i\right) \wedge \left(\bigwedge_{y_j \in V_y} \neg y_j\right)$;
9: (result, I_c) := Solve(Ψ_c) ;
10: **if** result = `UNSAT` **then** $\Sigma := \Sigma \cup \{I\}$;
11: $\Psi := \Psi \wedge \left(\bigvee_{x_i \in V_x} \neg x_i\right)$;

end

Figure 2.5: Minimal Model Generation Procedure [86]

$x_2 \wedge \ldots \wedge x_i) \wedge \neg y_1 \wedge \neg y_2 \wedge \ldots \wedge \neg y_j$ is unsatisfiable, where $I \cap V_p = \{x_1, x_2, \ldots, x_i\}$, $\overline{I} \cap V_p = \{y_1, y_2, \ldots, y_j\}$.

For instance, suppose that Ψ is the same one as the previous example, that is, $\Psi = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3)$. Besides, suppose we want to detect whether a model $\{x_2, x_3\}$ is minimal or not. Then we try to check the satisfiability of $\Psi_c = \Psi \wedge (\neg x_2 \vee \neg x_3) \wedge \neg x_1$. The formula Ψ_c will be `UNSAT`, and thus it is detected as a minimal model. In contrast to the minimal model $\{x_2, x_3\}$, the model $\{x_1, x_3\}$ is detected not to be a minimal model by this procedure.

Koshimura *et al.* also report a minimal model generator based on a SAT solver by utilizing the above theorem (see Figure 2.5). The procedure receives a CNF formula Ψ and a set of variables V_p as inputs. At Line 3 and 9, a function Solve corresponds to a

SAT solver that generates whether a given formula is satisfiable (`SAT`) or unsatisfiable (`UNSAT`) and a model I if the formula is satisfiable. In the iteration (Line 2-11), Solve is invoked with Ψ at first. If the result is `UNSAT` then a set of models Σ is returned (Line 4). Otherwise, Solve is invoked with Ψ_c to check whether the obtained model I is a minimal model or not (Line 9). If it is minimal model then the model is added to Σ (Line 10). Following that, the formula Ψ is updated with a block clause and resume procedures from the top of the iteration. This is continued until all minimal models are generated. Minimal model generation can be applied to real world problems and it actually very useful. We show an instantiation of such an application in Chapter 5.

Chapter 3

Incremental Satisfiability Solving

In this chapter, we study an approach to solve optimization and enumeration problems by multiple execution of a SAT solver, which we call incremental SAT solving. We give its general framework: it incrementally computes the satisfiability and models of propositional formulas until a given goal condition is satisfied. We also show how learned clauses are reused to accelerate incremental SAT solving.

3.1 Architecture of the Solving Method based on SAT Technologies

In this thesis, our objective is roughly to solve a variety of problems by utilizing SAT technologies. A natural architecture for this objective is shown in Figure 3.1. In the figure, the original problem is first encoded into a propositional formula. The formula is then passed to a SAT solver. As we mentioned in the preceding chapter, almost all SAT solvers can generate a model when a given formula is satisfiable. They thus can be used as a model generator. If the formula is satisfiable, then a model is (or models are) generated and we obtain a solution (or solutions) of the original problem by decoding; otherwise, no solution is returned. This architecture has an advantage

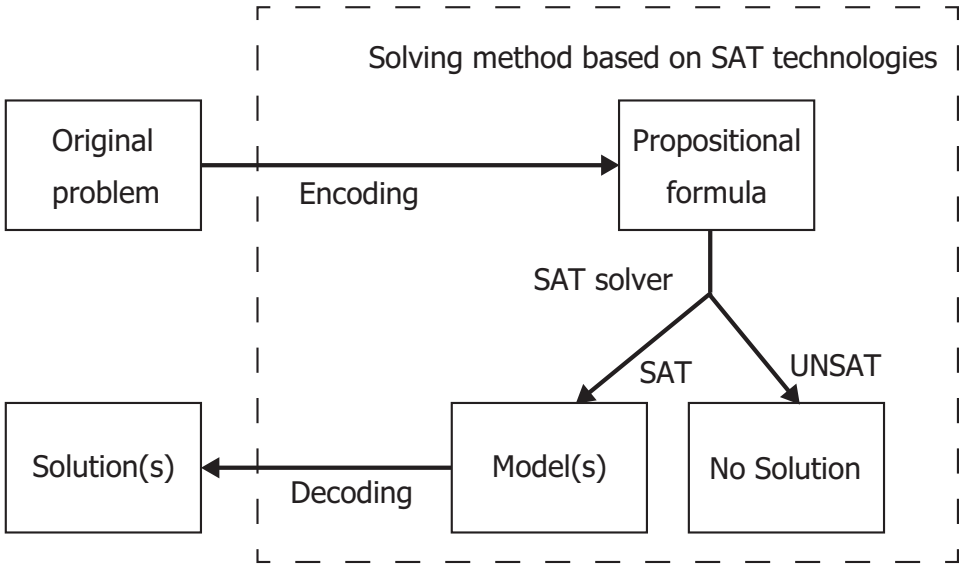

Figure 3.1: Solving Method based on SAT Technologies

that its core procedure is replaceable. It always allows us to employ the state-of-the-art SAT solvers. Moreover, we can follow its progress without modification of other parts of the architecture.

In this architecture, there are two technical issues should be considered. The first one is encoding from the original problem into a propositional formula. Many propositional encoding methods have been studied [136, 76, 55, 52, 127, 129, 130] and are available for various kinds of constraints. However, applications of the architecture are still limited only with these encoding methods. Thus, the second issue is to extend the architecture to break this limitation. In the following, we focus on a method that incrementally executes a SAT solver, which enables SAT technologies to be applied to a more wide range of problems such as optimization and enumeration problems.

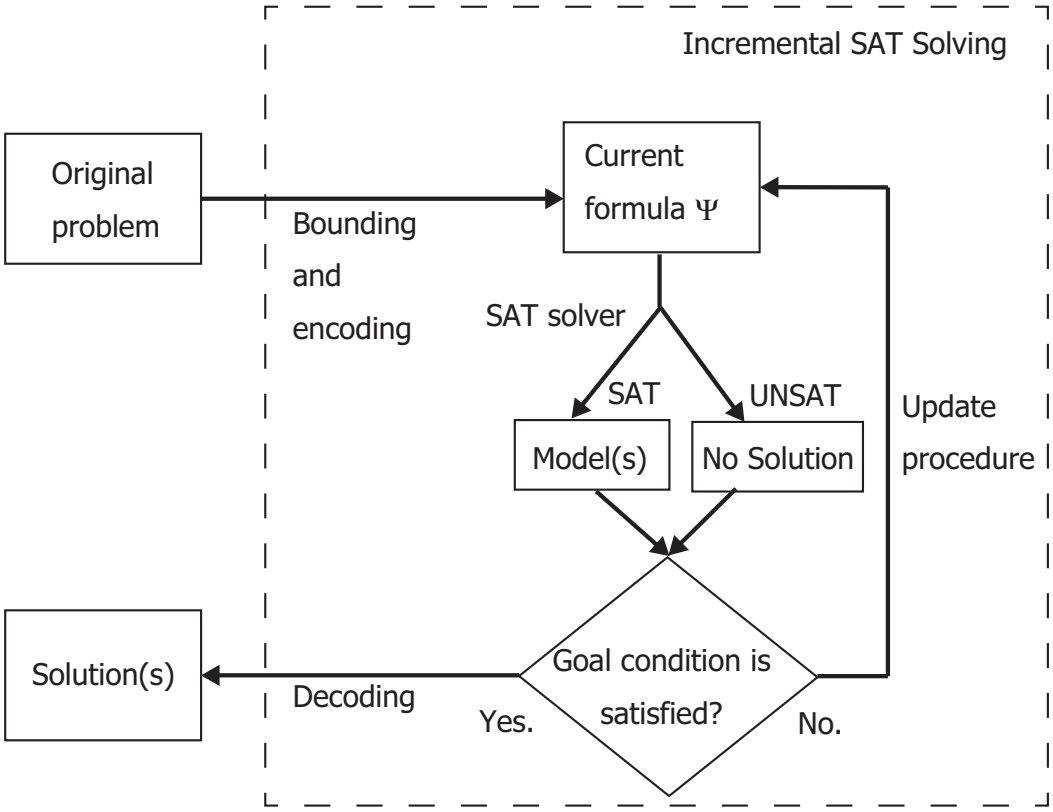

Figure 3.2: Incremental SAT Solving

3.2 Incremental Satisfiability Solving

In this section, we give the explanation of the incremental satisfiability solving (incremental SAT solving), which incrementally computes the satisfiability and models of propositional formulas until a given goal condition is satisfied. We can capture the incremental SAT solving is an extension of the solving method based on SAT technologies shown in Figure 3.1, which encodes the original problem into a propositional formula. We incrementally use that method to compute the solution of a given original problem. We show the architecture of incremental SAT solving in Figure 3.2. The flow of the incremental SAT solving is as follows: (I) at first, the method receives an *initial formula* Ψ_{init} encoded from a bounded original problem and substitutes the initial formula into the *current formula* Ψ. (II) then, the method computes the satis-

fiability and models of Ψ by a model generator; precisely, it depends on the original problem whether we need to obtain both satisfiability and a model (or models), or not. (Ⅲ) we then checks whether our *goal condition*, which is typically the current formula is `SAT` (or `UNSAT`), is satisfied or not. (Ⅳ) if it is satisfied then the method generates outputs, otherwise the current formula is converted by an *update procedure*. The update procedure constructs an updated formula from the current formula Ψ by adding a set of clauses ψ^+ and removing a set of clauses ψ^-. That is, the current formula Ψ is updated to $\Psi \cup \psi^+ \setminus \psi^-$. The procedure from (Ⅱ) to (Ⅳ) is incrementally continued until the goal condition is satisfied. Note that, in the process (Ⅱ), we do not keep all computed models in most cases. Typically, we only need the model of the current formula when the goal condition is satisfied. In the case of solving optimization problems, computing the satisfiability is enough to compute the optimal value. If we also need optimal solution then its model is used for decoding.

The architecture shown here is so general that it can solve a wide range of problems. Note that, since there is no limitation for the update procedure, we could have no intersection between the current formula and its updated one. However, considering applications, it is natural to assume that $\Psi \cap \Psi' \neq \emptyset$ holds and there is not so large changes, where Ψ and Ψ' are the current formula and its updated one. In addition, the update procedure is interchangeable with a manual procedure such as human operations. This is useful property when we embed the incremental SAT solving method into human interaction system e.g. graphical user interface. From the next section, we explain more detail of incremental SAT solving. In particular, we focus on the following three features to characterize several approaches of incremental SAT solving: (1) the initial formula, (2) the update procedure and (3) the goal condition.

3.3 Stepwise Approach to Optimization Problems

In this section, we first explain the *stepwise approach* of incremental SAT solving: it constructs a sub-problem by bounding the original problem and incrementally solve sub-problems with increasing its bound until a solution is found. We characterize this approach as follows:

(A1) Initial Formula. A propositional formula encoded from a problem bounded by the initial bound i.

(A2) Update Procedure. Convert the current formula to the formula that corresponds to the problem bounded by $i + 1$.

(A3) Goal Condition. The current formula is `SAT`.

As we reviewed in Section 1.1.3, several methods using SAT technologies to solve planning, bounded model checking and job-shop scheduling problems have been proposed [81, 19, 32, 43, 125, 74]. In the following, we show how the approaches for planning [81], bounded model checking [19], and scheduling problem [125, 74] are represented in this stepwise approach of incremental SAT solving.

Planning problem [81]. Kautz and Selman proposed a method that solves the classical planning problem by encoding it into SAT problems [81]. Following their studies, there are several researches studying this subject [82, 74, 104].

By using the plan length as the bound, the incremental SAT solving for this problem is considered as the following: (A1) the initial formula corresponds to the original planning problem bounded by the initial bound i. That is, this formula corresponds to the problem of deciding whether there is a plan of the length i or not. (A2) the update procedure converts the current formula to the next formula corresponding to the problem bounded by the incremented plan length $i + 1$. (A3) the goal condition is that the current problem is `SAT`. When the goal condition is firstly satisfied, there

must be a satisfiable formula whose model represents the shortest plan of the original problem.

Suppose that there is a planning problem that has the shortest plan length 4. One way to solve this problem is as follows. We construct an initial formula Ψ_1 corresponding to the planning problem bounded by the plan length 1. The satisfiability of Ψ_1 is then computed and we suppose that `UNSAT` is returned. Then, the update procedure converts Ψ_1 to Ψ_2. This process is repeated such as Ψ_1, Ψ_2, Ψ_3, ..., and suppose that Ψ_4 is firstly decided as `SAT`. Then, the plan length 4 is found as the shortest plan of this planning problem and we can obtain the actual plan of this length by decoding the obtained model of Ψ_4.

Job-shop Scheduling Problem [32, 74]. Crawford and Baker proposed a method for the decision version of the job-shop scheduling problems that is a problem of deciding whether all jobs can be done by a given makespan or not [32]. They encode this decision problem into a SAT problem. However, they did not consider the optimization version of the job-shop scheduling problem that is an optimization problem of computing the optimal makespan. Inoue *et al.* studied their approach and show how the optimal value is computed by finding the boundary of the satisfiability [74].

By using the makespan as the bound, the incremental SAT solving for this problem is represented as follows: (A1) the initial formula corresponds to the original job-shop scheduling problem bounded by the initial makespan i, which is typically its lowest bound. That is, this formula corresponds to the problem of deciding whether there is an ordering of operations that can complete all jobs by the makespan i. (A2) the update procedure converts the current formula to the next current formula corresponding to the problem bounded by the incremented makespan $i + 1$. In other words, the updated procedure converts the current problem to the next one of an incremented makespan. (A3) the goal condition is that the current problem is `SAT`.

When the goal condition is firstly satisfied, there must be a satisfiable formula whose model represents an ordering that can complete all jobs in the shortest makespan.

There are several approaches solving this problem by the bisection method [125, 74, 104]. We will explain this bisection method in a later section.

Bounded Model Checking [19]. Biere *et al.* proposed a method for verifying reactive systems and they particular focus on symbolic model checking [19]. They represent the original problem in linear temporal logic (LTL) and encode LTL into propositional formulas. Their basic ideas is to consider a path of a finite length k and identify a counter example. They call this method as *bounded model checking* (BMC). In BMC, they bound the original model chocking problem by path length k.

The incremental SAT solving for this problem is represented as follows: (A1) the initial formula corresponds to the original model checking problem bounded by the initial length k. That is, this formula corresponds to the problem of deciding whether there is a counter example of length k or not. (A2) the update procedure converts the current formula to the next current formula corresponding to the problem bounded by the incremented path length $k+1$. In other words, the updated procedure converts the current problem to the next one that has a counter example of an incremented length. (A3) the goal condition is that the current problem is `SAT`. When the goal condition is firstly satisfied, there must be a satisfiable formula whose model represents the shortest counter example of the original problem. In addition to the research in [19], Eén and Sörensson proposed to apply the incremental SAT solving to temporal induction [43] and they also reported how to utilize learned clauses.

In the above, we show that how incremental SAT solving is applied to problems such as planning, job-shop scheduling problem and bounded model checking. Among them, the job-shop scheduling problem is a typical optimization problem. It means that we can apply incremental SAT solving to optimization problems.

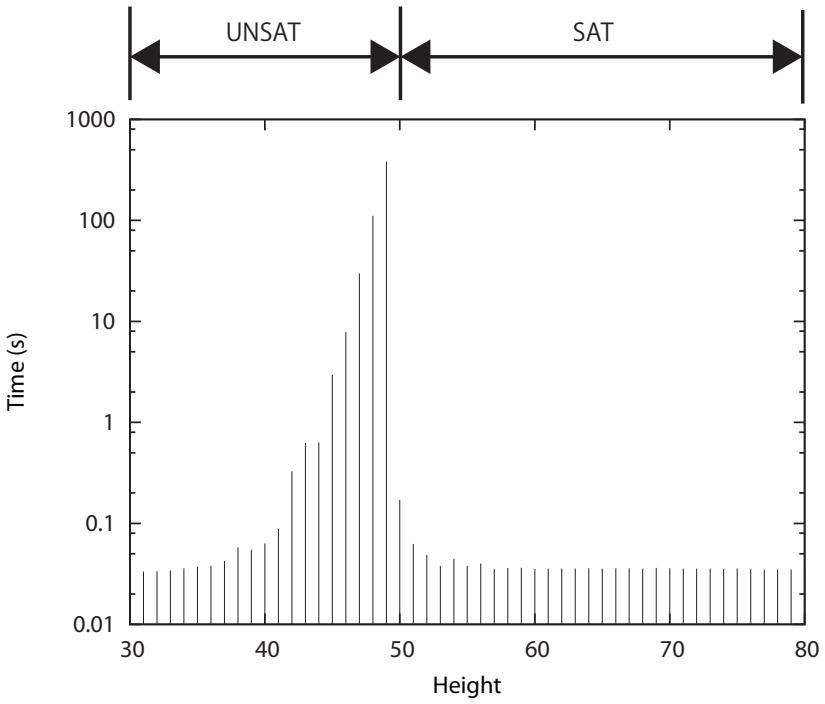

Figure 3.3: Computation Time around the Optimal Value on 2SPP

Optimization problems are a very important research subject in Operations Research, Constraint Programming and Artificial Intelligence. Tremendous numbers of research have been done for this subject. There are well known problems, such as the job-shop scheduling problem, the traveling salesman problem, the strip packing problem and the knapsack problem, just to name a few. In the following, we explain three ways to solve optimization problems by incremental SAT solving. One way is approaching the optimal value from the lowest bound, which is a standard way to reach the optimal value and same as the stepwise approach of the incremental SAT solving described above. Thus, in solving this problem, we start to solve Ψ_{lb} and continue to solve $\Psi_{lb}, \Psi_{lb+1}, \ldots$ until some Ψ_i is firstly decided to `SAT`. Simultaneously, the bound i is proved as the optimal value.

As we have seen, we can bound an optimization problem and divide it into multiple sub-problems such that each of them has each fixed bound. For instance, suppose that

an optimization problem that has 4 as a lowest bound and 12 as an upper bound. In this case, we first bound the original optimization problem into a problem bounded by 4. Obviously, an encoded formula from this problem is `SAT` if the bounded problem has a solution and `UNSAT` otherwise. Suppose that, we obtained the satisfiability of bounded problems as follows: `UNSAT` at 4, `UNSAT` at 5, `SAT` at 6, `SAT` at 7 and so on. Then, the optimal value of the original problem is proved as 6. That is, the optimal value exists at the bound between `UNSAT` and `SAT`. As a practical example of the computation of incremental SAT solving, Figure 3.3 shows an experimental result for an optimization problem instance of the two-dimensional strip packing problem (2SPP), which we will detail in Chapter 4. The graph shows computation time for each height that is the value to be optimized, and the height of 50 is the optimal value for this instance. In these experiments, we invoke a SAT solver from scratch for each instance.

As we can read from the figure, almost all problems of larger height than the optimal value, which are `SAT` problems, are easier. In contrast to `SAT` ones, problems of lower height than the optimal value, which are `UNSAT` problems, become difficult as the value is getting close to the optimal value. An interesting point is that computational time of the problem of the height of 49 and 50 are dramatically different. The one reason of the difference is that we need to search larger space to prove unsatisfiability of the height of 49 because this problem is almost satisfiable.

Though there is slightly a difference, many optimization problems basically follow this property: bounded problems closed to optimal value, especially unsatisfiable problems, are difficult to solve. Thus, the stepwise approach described in the above is not always appropriate to reach the optimal value because it needs to solve a number of unsatisfiable problems. We thus explain two more approaches for optimization problems in addition to the stepwise approach of incremental SAT solving.

One is approaching the optimal value from the highest bound, which is called the

decremental approach. This is the reverse way of the stepwise approach of incremental SAT solving. There is an optimization problem that has a bound value i whose highest bound is ub. Then, the decremental approach of incremental SAT solving is constructed as follows:

(B1) Initial Formula. A propositional formula $\Psi_{i=ub}$ encoded from an optimization problem bounded by the bound ub.

(B2) Update Procedure. Convert the current formula to the formula that corresponds to the problem bounded by the bound $i-1$.

(B3) Goal Condition. The current formula is `UNSAT`.

In solving this problem, we start to solve Ψ_{ub} and continue to solve $\Psi_{ub}, \Psi_{ub-1}, \ldots$ until the satisfiability of Ψ_i is decided to `UNSAT`. If the satisfiability of Ψ_i is `UNSAT`, then the value $i+1$ is proved as the optimal value.

The third way is approaching to the optimal value by the *bisection method*. If the lower bound lb and the upper bound ub are known in advance, we can use the bisection method. It reaches the optimal value to iteratively reduce the possible value region by half. In other words, it chooses a half value of the range between a current lower bound and a current upper bound. The purpose of the bisection method is to approach the bound of `SAT` and `UNSAT` as fast as possible. The update procedure takes the satisfiability of Ψ_i as an input, and then it proceeds the following calculation:

$$(lb, ub) = \begin{cases} (lb, i) & \Psi_i \text{ is } \texttt{SAT} \\ (i+1, ub) & \Psi_i \text{ is } \texttt{UNSAT} \end{cases}$$

In the next, the region is split using updated bounds as $i' = \lfloor (lb + ub)/2 \rfloor$ and the updated $\Psi_{i'}$ is returned. The goal condition is given as that the current lower bound lb and upper bound ub do not satisfy $lb < ub$.

Bisection method (Initial Formula Ψ, Lower Bound lb, Upper Bound ub)

begin

1: **while** $lb < ub$

2: $o := \lfloor (lb + ub)/2 \rfloor$;

3: $\Psi := \Psi \cup \psi_o$;

4: $(result, \psi) :=$ solve (Ψ);

5: **if** $result$ is `SAT` **then** $ub := o$;

6: **else** $lb := o + 1$;

7: $\Psi := \Psi \setminus \psi_o$;

8: **return** ub;

end

Figure 3.4: Bisection Method

(C1) Initial Formula. A propositional formula Ψ_i encoded from a problem bounded by a bound i such that $i = \lfloor (lb + ub)/2 \rfloor$.

(C2) Update Procedure. Convert the current formula Ψ_i to the formula $\Psi_{i'}$ that corresponds to the problem of the value i'.

(C3) Goal Condition. The current bounds do not satisfy $lb < ub$.

An algorithmic representation of this procedure is shown in Figure3.4. The algorithm consists of an iterative procedure. The iteration will continue until the bound between `SAT` and `UNSAT` is found, which is shown as $lb < ub$ in the algorithm (Line 1). In the iteration, it splits the range by half and o will be the half value of the current range (Line 2). It then invokes a function solve, which corresponds to a model generator, with a CNF formula $\Psi \cup \psi_o$ representing a problem with the value o (Line 3 and 4). The function solve then returns the result of satisfiability of the given CNF formula. Then, the result is checked. If it is `SAT`, then the upper bound of the range is

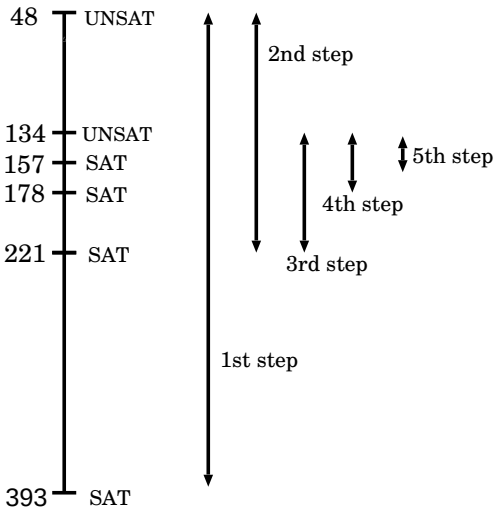

Figure 3.5: Bisection Method [74]

updated to o. Otherwise, the lower bound is updated to $o + 1$. Finally, the algorithm returns the latest upper bound as the optimal value. Note that the latest upper bound after the iteration indicates the lowest value that is satisfiable.

This bisection approach of the incremental SAT solving is studied and applied to the job-shop scheduling problem in [125, 74, 104]. For instance, suppose that there is a job-shop scheduling problem whose upper and lower bounds are respectively set to 48 and 393. By the bisection approach, the bounded makespan is changed as: 48 (`UNSAT`), 393 (`SAT`), 221 (`SAT`), 134 (`UNSAT`), 178 (`SAT`), 157 (`SAT`), ..., and the range of the optimal makespan is getting small (see Figure 3.5).

3.4 Stepwise Enumeration

Slightly different to these problems in the previous section, we sometimes want to have more or all solutions rather than single solution. In particular, there are several applications of enumeration of all solutions [99, 78]. Since the core procedure of incremental SAT solving is a model generator as is shown in Figure 3.2, we can enumerate

all solutions in each bounded problem. This stepwise enumeration exploits interesting problem domains such as biological problems. In such problems, it is sometimes not adequate to generate single solution on the smallest bound and we want to continue to enumerate solutions until a solution in which we are interested is found. We call this solution as a *preferred solution*. Besides, we call the model decoded to a preferred solution as the *preferred model*. In this case, the goal condition is that the preferred solution is found instead of just obtaining `SAT` (or `UNSAT`). This stepwise enumeration of the incremental SAT solving is constructed as follows:

(D1) Initial Formula. A propositional formula encoded from a problem bounded by the initial bound i.

(D2) Update Procedure. Convert the current formula to the formula that corresponds to the problem bounded by $i + 1$.

(D3) Goal Condition. The preferred model is found in computed models.

We will show the specific application of this stepwise enumeration in Chapter 5.

3.5 Reusing Learned Clauses

In this section, we show how to utilize learned clauses to accelerate the incremental SAT solving. As we described in Section 2.2.2, CDCL-solvers can avoid a same conflict in its search by learned clauses. Since there is at most the difference of a few clauses between the current formula and its updated formula in most cases, if it is possible to reuse learned clauses to the next search, we can avoid redundant conflict occurred while computing previous formulas. In the following, we explain a condition of learned clause reusability that is formally confirmed by Nabeshima *et al.* [104].

We follow their explanation using Figure 3.6. The figure shows any conflict graph constructing a conflict between literals l and $\bar{l}$. Let ψ be a CNF formula and c be the

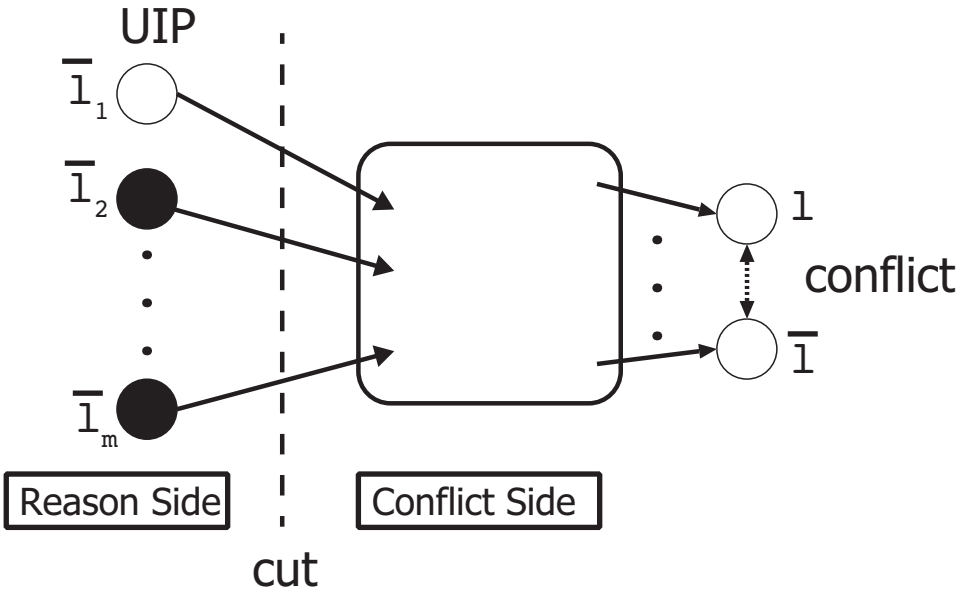

Figure 3.6: Abstracted Conflict Graph [104]

learned clause generated from this conflict graph. In the figure, $l_1, \ldots, l_m$ represent literals such that $c = l_1 \vee \cdots \vee l_m$. The white node denotes a literal $\overline{l_1}$ implied in the current decision level and it is also an UIP. The Black nodes denote literals that are implied earlier than the white node in all decision levels. The white box contains nodes assigned after $\overline{l_1}$. Let $D_1, \ldots D_n$ be the set of clauses used for constructing this conflict graph. In the following, we introduce three mappings S, T, U such that $D_i = \bigvee_{l_a \in S(D_i)} l_a \vee \bigvee_{l_b \in T(D_i)} l_b \vee U(D_i)$. Specifically, those are defined as follows. Let S be a mapping such that $S(D_i)$ representing a set of literals in the clause D_i, each of which is either the UIP literal l_1 or a literal implied after l_1. That is, for $l_j \in S(D_i)$, $\overline{l_j}$ must be an elements of the conflict graph and these are included in the white box or the complement of the UIP literal. Obviously, $S(D_i) \neq \emptyset$ holds. Let T be a mapping such that $T(D_i)$ representing a set of literals in the clause D_i that are implied earlier than l_1. That is, $T(D_i)$ is a set of literals assigned to $False$ before this conflict graph is made and every clause in $T(D_i)$ is a black node in the above conflict graph. Let U

be a mapping such that $U(D_i)$ represents the literal that is lastly assigned in a clause D_i. That is, $l_j = U(D_i)$ must be an elements of the conflict graph and it is included either in the white box or the conflict literals. We then have the following properties.

Proposition 1 (Nabeshima *et al.* [104]). Let $D_1, \ldots, D_n$ be clauses constructing the above conflict graph. Then, the length of any clauses D_i is two or more.

Proof. Since $U(D_i) = l_j$, $S(D_i) \neq \emptyset$ and $l_j \notin S(D_i)$, each clause D_i has at least two literals. □

Theorem 2 (Nabeshima *et al.* [104]). Let Ψ be a CNF formula and c be any learned clause generated while solving Ψ. Then c is a logical consequence of a set of some non-unit clauses in Ψ.

Proof. Let $D_1, \ldots, D_n$ be clauses in Ψ constructing the conflict graph shown in Figure 3.6, which causes a conflict $l \wedge \bar{l}$ and generates a learned clause $c = l_1 \vee \cdots \vee l_m$. We show that any model of $D_1 \wedge \cdots \wedge D_n$ satisfies c, that is $D_1 \wedge \cdots \wedge D_n \models c$, through a proof by contradiction. We assume that there exists a model I of $D_1 \wedge \cdots \wedge D_n$ that does not satisfy $c = l_1 \vee \cdots \vee l_m$. That is, every literal l_i $(1 \leq i \leq m)$ is assigned to $False$ by I. Thus, I satisfies $\overline{l_1} \wedge \cdots \wedge \overline{l_m}$. However, if $\overline{l_1} \wedge \cdots \wedge \overline{l_m}$ is true, then it immediately causes the conflict $l \wedge \bar{l}$ by the conflict graph. Thus, I cannot be a model of $D_1 \wedge \cdots \wedge D_n$ and it contradicts the assumption. Hence, $D_1 \wedge \cdots \wedge D_n \models c$ holds. Therefore, since the length of each clause of $D_1, \ldots, D_n$ is more than two by Proposition 1, c is a logical consequence of a set of non-unit clauses $\{D_1, \ldots, D_n\}$ included in Ψ. □

Finally, we obtain the reusability condition by Nabeshima *et al.* as follows.

Proposition 2 (Reusability Condition by Nabeshima *et al.* [104]). Let Ψ and Ψ' be CNF formulas. Then, any leaned clauses generated while solving Ψ are reusable for solving Ψ' if the following condition holds: Ψ' includes all non-unit clauses of Ψ.

Proof. It can be proved from Theorem 2.

By this condition, we can reuse any learned clauses generated while solving Ψ to solve Ψ'. Thus, we can avoid to re-search meaningless search space that is confirmed while solving Ψ. Eén and Sörensson also mentioned the above condition and proposed an interface of a SAT solver that only allows removals of unit clauses [42]. In their implementation, those unit clauses are added during particular single solving, and then automatically removed from the solver instance. By this restriction of the removal, they indicated that all learned clauses are not needed to remove whenever some clauses are removed from formulas. This feature is implemented in a SAT solver Minisat [42].

3.6 Related Work

In the research domain of the constraint satisfaction problem (CSP), there are several researches that study a problem of solving a sequence of CSPs [37, 50, 100, 88, 46]. This kind of problem is referred to as *dynamic constraint satisfaction problem* (DCSP), whose notion is first introduced by Dechtor and Dechtor [37]. They consider a sequence of constraint networks that each one constructed from a change in the preceding one. They showed that such changes between two networks are represented in addition or removal of constraints and variables. Following that, Freeman-benson proposed the DeltaBlue algorithm that can handle the removal and the addition of constraints [50]. Their algorithm uses the current solution as a guide to finding the next one rather than starting from scratch. Mittal and Falkenhainer proposed a variant of DCSP [100]. In their definition, the changes between two CSPs are given by the appearance of optional variables. Besides, those optional variables are controlled by *activity constraints* that are a function of the current assignment. Other variant of DCSP called *open constraint satisfaction problem* is proposed in [88, 46]. In their definition, the changes of values in variable domains as well as constraints are allowed.

In addition to the above, there are more variants of DCSP. These are well summarized in the literature [135]. In contrast to those researches, we focus on the solving method rather than formalization of problems to apply SAT technologies to optimization and enumeration problems. Although our attention is on limited domain compared with DCSP, the motivation to apply the state-of-the-art SAT techniques to a more wide range of problems is important.

Hooker proposed a notion of *incremental satisfiability* (incremental SAT) problem [70] as follows: given that a set Ψ of propositional clauses that is satisfiable, check whether $\Psi \cup \{c\}$ is satisfiable for a given clause c. He proposed a variant of the DPLL procedure that is adapted to incrementally solve multiple problems [70]. Bennaceur *et al.* slightly generalized Hooker's definition to allow to add a set of clauses [18] and proposed an incremental branch-and-bound method, which includes Lagrangean relaxations, meta-heuristics, and judicious jumping back [18]. In contrast to those researches, the update procedure of the incremental SAT solving allows both addition and removal of clauses. Besides, we consider that the next formula is not given until the computation for the current formula ends.

Whittemore *et al.* reported a solver called SATIRE that is intended to solve the sequence of CNF formulas [140]. In addition to CNF formulas, it can treat pseudo-Boolean constraints. The solver SATIRE is based on CDCL SAT solvers. Their idea is also to reuse learned clauses generated while solving previous problems. Although they allow removal of any clauses, they need to record clauses that are responsible for each learned clause. In our case, we can reuse any learned clauses without any concern by Proposition 2.

3.7 Summary

SAT technologies have been applied to several kinds of problems such as bounded model checking, planning and scheduling [19, 81, 32, 125, 74], which are shown in this chapter. These methods convert their original problems into SAT problems and compute the satisfiability of each problem to solve the original problems. In this chapter, we review their approaches and show how those approaches are represented in incremental SAT solving and how it is applied to optimization and enumeration problems. In addition, we show a condition of the reusability of learned clauses, which is shown in [104], to effectively solve problems. In the following two sections, we apply the incremental SAT solving to two problems. One is an optimization problem called *two-dimensional strip packing problem*. In solving this problem, we also show the effectiveness of reusing learned clauses. The other one is an enumeration problem called *minimal active pathway finding problem*. We propose this problem to analyze metabolic pathways in Systems Biology and show how incremental SAT solving is applied to this problem.

Chapter 4

Two-dimensional Strip Packing Problem

In this chapter, we give a specific application of SAT technologies. We apply incremental SAT solving to the two-dimensional strip packing problem (2SPP), which has been studied in Operations Research. We give a propositional encoding of the problem and compare our method with state-of-the-art ad-hoc methods of 2SPP.

4.1 Introduction

Packing problems have many practical applications such as truck loading, LSI layouts and assignments of newspaper articles [41, 68, 113]. There has been a great deal of research on these problems, for instance, knapsack problems and bin packing problems. In this chapter, we consider a SAT-based exact method for *the two-dimensional strip packing problem* (2SPP) [10, 93]. This problem is NP-hard in the strong sense, because the one-dimensional bin packing problem, which is strongly NP-hard, can be transformed into a 2SPP [51].

The input of the 2SPP is a set $R = \{r_1, \ldots, r_n\}$ of n rectangles. Each rectangle

has width w_i and height h_i. We are also given a large rectangle, called a *strip*, of width W. The goal is to pack all rectangles without overlapping into the strip by minimizing the height H of the strip. Although rectangles are allowed to be rotated 90 degrees in several real-world applications, we assume that the rectangles cannot be rotated according to the convention of previous research [97, 3, 138, 4]. Furthermore, we assume that only integer values are allowed for w_i, h_i, W, and H.

The 2SPP has been extensively studied in the last decade. There are two main ways to solve it: *incomplete* and *exact*. Many studies have focused on incomplete methods [3, 138, 25, 105]. Recently in 2008, Alvarez-Valdes *et al.* [3] applied the GRASP algorithm [31] to the 2SPP; the GRASP algorithm is an iterative procedure combining a constructive phase and an improvement phase. Their method can pack over thousand rectangles while keeping the quality of the solutions. However, incomplete methods cannot prove the optimality of the solution, *i.e.*, it cannot decide whether the computed height is minimum or not. Incomplete methods can confirm that a solution is optimal only when the solution corresponds to the lower bound of the problem. On the other hand, the exact method can compute the optimal solution of the problem. In 2003, Martello *et al.* [97] proposed a branch and bound algorithm and reported good lower bounds produced from a relaxation of the problem. They applied these bounds in their exact procedure and obtained results on 38 instances. In 2008, Alvarez-Valdes *et al.* proposed hybrid method contains both a branch and bound method and an incomplete method [4]. Their hybrid method tries to narrow down the range of optimal heights with the incomplete method, and if the last solution does not correspond to the lower bound, the hybrid method executes a branch and bound method. Although both complete and incomplete methods have been well studied, it is still difficult to compute the optimal height even for small problems consisting of less than 200 rectangles.

To solve such hard problems, we propose a SAT-based method. As far as the

authors are aware, this is the first article about solving the 2SPP with a SAT solver. One advantage of our method is that it can utilize several SAT techniques which have been actively studied in recent years [58]. Most of *SAT solvers* are based on the Davis–Putnam–Logemann–Loveland (DPLL) algorithm [34] and use heuristics such as backjumping, variable selection, restart, and *clause learning* [94, 102, 72]. For instance, using clause learning, SAT solvers generate learned clauses when a conflict occurs, and it avoids the same conflict later in the search. These techniques make SAT solvers applicable to huge SAT problems consisting of over millions of clauses.

In order to solve the 2SPP with a SAT solver, we represent the problem as a *constraint satisfaction problem* (CSP) and solve the CSP through a SAT encoding called *order encoding* [127]. This encoding is a generalization of the one used to encode the job-shop scheduling problems by Crawford and Baker [32]. In order encoding, a comparison $x \leq a$ is encoded into a Boolean variable. For the 2SPP, it enables more compact encoding compared with other encodings such as direct encoding. Still, the translated SAT problems quantitatively grow with the number of input rectangles. Therefore, we propose five techniques to reduce the search space. Two of them break symmetries and others utilize relations between rectangles to solve the 2SPP more efficiently. To compute the optimal solution, we apply incremental SAT solving to the 2SPP. We thus reuse the learned clauses generated from the previously solved problem. We also reuse assumptions that restrict the height of the 2SPP. For evaluation, we made three comparisons for 38 instances from the literature [97]. First, we made a comparison with the state-of-the-art CSP solver. Second, we made a comparison between our own reduction and reuse techniques. Third, we made a comparison with the branch and bound method proposed by Alvarez-Valdes [4] and other method by Kenmochi *et al.* [83].

This chapter is based on the literature by Soh *et al.* [126], and contains new materials including wider comparison with other methods and extensive discussion on SAT

encoding methods. The remainder of the chapter is organized as follows. Section 4.2 provides preliminaries on the 2SPP and related concepts. Section 4.3 describes how to encode a decision-version of the 2SPP into SAT. Section 4.4 explains how to compute the optimal height of a 2SPP and several techniques to solve problems more efficiently. Section 4.5 shows computational results. Section 4.6 discusses the effectiveness of our method and Section 4.7 introduces related work. Section 4.8 concludes this chapter.

4.2 Preliminaries

In this section, we give preliminaries to present a SAT-based approach to solving the 2SPP. In the following, $\mathbb{N}$ denotes the set of natural numbers and $\mathbb{Z}$ denotes the set of integers.

4.2.1 2SPP and 2OPP

We consider a SAT-based approach to solving the *two-dimensional strip packing problem* (2SPP). Although we want to obtain the optimal height of the 2SPP, SAT solvers can only determine the satisfiability of a given problem. We thus bound the 2SPP and obtain *two-dimensional orthogonal packing problems* (2OPPs), which is a decision-version of the 2SPP. The difference between the 2SPP and the 2OPP is that a strip height is given in advance in the 2OPP. We describe the detail of how to give such a fixed height in Section 4.4.1. We here define the 2SPP and the 2OPP as follows [10].

Definition 7 *Two-dimensional strip packing problem (2SPP)*

Input. A set $R = \{r_1, \ldots, r_n\}$ of n rectangles, where each rectangle $r_i \in R$ has width w_i and height h_i ($w_i, h_i \in \mathbb{N}$), and a *strip* of width $W \in \mathbb{N}$.

Constraint. Each rectangle cannot overlap with the others and the edges of the strip, and the rectangles are parallel to the horizontal and vertical axes.

Question. What is the minimum height such that the set of rectangles can be packed in the given strip?

Definition 8 *Two-dimensional orthogonal packing problem (2OPP)*

Input. A set $R = \{r_1, \ldots, r_n\}$ of n rectangles, where each rectangle $r_i \in R$ has width w_i and height h_i ($w_i, h_i \in \mathbb{N}$), and a *strip* of width W and height H ($W, H \in \mathbb{N}$).

Constraint. Each rectangle cannot overlap with the others and the edges of the strip, and the rectangles are parallel to the horizontal and vertical axes.

Question. Can the set of rectangles be packed in the given strip?

We represent the 2OPP as a *constraint satisfaction problem* (CSP). A CSP is a triple $\langle V, D, C \rangle$, where V is a finite subset of integer variables and D is a function that maps every variable in V to a subset of $\mathbb{Z}$. We use $D(x)$ as the subset of $\mathbb{Z}$ mapped from $x \in V$ and call the set $D(x)$ the domain of x. C is a finite set of constraints over a set of variables in V.

A CSP formulation of the 2OPP is as follows. Let x_i and y_i be integer variables such that the pair (x_i, y_i) of variables represents the position of lower left coordinates of the rectangle r_i in the strip (see Figure 4.1). The domains of x_i and y_i are as follows.

$$\begin{aligned} D(x_i) &= \{a \in \mathbb{N} \mid 0 \leq a \leq W - w_i\} \\ D(y_i) &= \{a \in \mathbb{N} \mid 0 \leq a \leq H - h_i\} \end{aligned} \tag{4.1}$$

These domains represent possible values of the coordinates of the rectangle r_i and guarantee that r_i must not overlap the edges of the strip. For each pair of rectangles r_i and r_j ($1 \leq i < j \leq n$), we associate the following *non-overlapping constraint*, which is introduced in the literature [96].

$$(x_i + w_i \leq x_j) \vee (x_j + w_j \leq x_i) \vee (y_i + h_i \leq y_j) \vee (y_j + h_j \leq y_i) \tag{4.2}$$

4.2.2 Order Encoding

There have been several studies on translations to encode CSPs into SAT problems. One is the direct encoding reported by Walsh [136]. In this encoding, if the CSP variable x is assigned to an integer constant c then the Boolean variable $p(x, i)$ must be *true*. Since this translation is natural and simple, direct encoding is widely used. Besides the direct encoding, there are many other studies such as support encoding [55], log encoding, log support encoding [52] and comparisons of those encodings in solving graph coloring problems [112].

There have also been many studies on translation methods that encode inequalities into SAT. Warners [137] reported a translation method for encoding inequalities into SAT and conducted experiments on the frequency assignment problem. In particular, that method was more efficient than the previous method using CPLEX. Bailleux and Boufkhad [7] and Sinz [124] reported a translation from cardinality constraints into SAT problems. Bailleux and Boufkhad [8] also showed an encoding from counting constraints into SAT problems. Eén and Sörensson [44] and Bailleux *et al.* [9] reported a translation from pseudo-Boolean into SAT.

Among them, *order encoding* by Tamura *et al.* [127] is a generalization of the one used to encode job-shop scheduling problems by Crawford and Baker [32]. It encodes a comparison $x \leq a$ by using a different Boolean variable for each integer variable x and integer value a. While direct encoding [136] represents constraints as points, order encoding can represent them as a single region. We discuss the differences between those methods in Section 4.6. Order encoding aims to make a more natural explanation of the order relation of integers.

There are two encoding steps in the order encoding. Let x be an integer variable, x_i's integer variables, a_i's non-zero integers, $c,\ d$ integers and $\ell,\ u$ mappings respectively to the lower and the upper bound. A *primitive comparison* is in the form of $x \leq d$. Let us consider the encoding for a constraint $\sum_{i=1}^{n} a_i x_i \leq c$ which has n integer

variables. In the first step, the constraint is translated into primitive comparisons as follows:

$$\bigwedge_{b_i} \bigvee_{i} (a_i x_i \leq b_i)^{\#} \tag{4.3}$$

where the parameters b_i's satisfy $\sum_{i=1}^{n} b_i = c - n + 1$ and $\ell(a_i x_i) - 1 \leq b_i \leq u(a_i x_i)$. The translation $()^{\#}$ is defined as follows.

$$(ax \leq b)^{\#} \equiv \begin{cases} x \leq \lfloor \frac{b}{a} \rfloor & (a > 0) \\ \neg(x \leq \lceil \frac{b}{a} \rceil - 1) & (a < 0) \end{cases} \tag{4.4}$$

In the second step, we translate the obtained primitive comparisons $x_i \leq c$ into new Boolean variables $p(x_i, c)$; that is, $x_i \leq c$ is satisfied if and only if $p(x_i, c)$ is $true$. We also need to introduce the following axiom clauses $A(x)$ for each integer variable x in order to represent the bound and the order relation.

$$A(x) = \{\{\neg p(x, \ell(x) - 1)\}, \{p(x, u(x))\}\} \cup \{\{\neg p(x, c-1), p(x, c)\} \mid \ell(x) \leq c \leq u(x)\} \tag{4.5}$$

Although $\{\neg p(x, \ell(x) - 1)\}$ is always $false$ and $\{p(x, u(x))\}$ is always $true$, we use a wider range here for ease of explanation.

Example. We illustrate order encoding with a simple constraint $x_1 + 1 \leq x_2$ ($x_1, x_2 \in \{0, 1, 2, 3\}$), which indicates a non-overlapping constraint between rectangles r_1 and r_2 shown in Figure 4.1. This constraint is encoded into the set of primitive comparisons as follows:

$$\neg(x_2 \leq 0), \quad (x_1 \leq 0) \vee \neg(x_2 \leq 1), \quad (x_1 \leq 1) \vee \neg(x_2 \leq 2), \quad (x_1 \leq 2)$$

These constraints are then translated into the following clauses:

$$\neg p(x_2, 0), \quad p(x_1, 0) \vee \neg p(x_2, 1), \quad p(x_1, 1) \vee \neg p(x_2, 2), \quad p(x_1, 2)$$

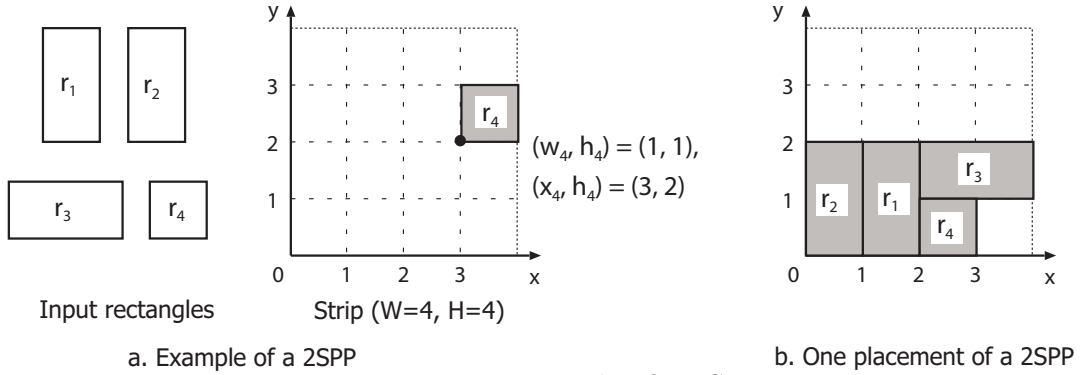

Figure 4.1: Example of a 2SPP

The following axiom clauses are also needed:

$$\neg p(x_1, 0) \vee p(x_1, 1), \quad \neg p(x_1, 1) \vee p(x_1, 2), \quad \neg p(x_2, 0) \vee p(x_2, 1), \quad \neg p(x_2, 1) \vee p(x_2, 2)$$

4.3 Encoding from 2OPP into SAT

In this section, we explain how to translate the 2OPP into a SAT problem with order encoding. Let $r_i, r_j \in R$ $(i \neq j)$ be two rectangles in a 2OPP. Let (x_i, y_i) and (x_j, y_j) be the lower left coordinates of the rectangles r_i and r_j, respectively. Let e and f be any integer. Then, the SAT encoding of a 2OPP uses the following Boolean variables: $lr_{i,j}$ is *true* if r_i is placed to the left of the r_j. $ud_{i,j}$ is *true* if r_i is placed below r_j. $p(x_i, e)$ is *true* if x_i is less than or equal to e. $p(y_i, f)$ is *true* if y_i is less than or equal to f.

Using those variables, we can encode constraints of the 2OPP into the clauses. For each rectangle r_i, and integer e and f such that $0 \leq e < W - w_i$ and $0 \leq f < H - h_i$, we have the 2-literal axiom clauses (4.5),

$$\begin{aligned} &\neg p(x_i, e) \vee p(x_i, e+1) \\ &\neg p(y_i, f) \vee p(y_i, f+1) \end{aligned} \tag{4.6}$$

For each rectangle r_i, r_j $(i < j)$, we have the following 4-literal clauses as the non-

overlapping constraints (4.2):

$$lr_{i,j} \vee lr_{j,i} \vee ud_{i,j} \vee ud_{j,i} \tag{4.7}$$

For each rectangle r_i, r_j $(i < j)$, and integer e and f such that $0 \leq e < W - w_i$ and $0 \leq f < H - h_i$[1], we also have the following 3-literal clauses that represent meanings of $lr_{i,j}$, $lr_{j,i}$, $ud_{i,j}$ and $ud_{j,i}$:

$$\begin{aligned}
&\neg lr_{i,j} \vee p(x_i, e) \vee \neg p(x_j, e + w_i) \\
&\neg lr_{j,i} \vee p(x_j, e) \vee \neg p(x_i, e + w_j) \\
&\neg ud_{i,j} \vee p(y_i, f) \vee \neg p(y_j, f + h_i) \\
&\neg ud_{j,i} \vee p(y_j, f) \vee \neg p(y_i, f + h_j)
\end{aligned} \tag{4.8}$$

Supposing that $W = H$, the number of clauses of a SAT-encoded 2OPP are $O(n^2)$, where n is the number of rectangles (see Figure 4.3).

Example. Consider the simple 2OPP which is obtained from a 2SPP shown in Figure 4.1a. We are given four rectangles $(w_1, h_1) = (1, 2)$, $(w_2, h_2) = (1, 2)$, $(w_3, h_3) = (2, 1)$, $(w_4, h_4) = (1, 1)$ and a strip $(W, H) = (4, 4)$. The result of the encoding is shown in Figure 4.2. A part of the feasible assignment of this SAT problem is as follows:

$$\begin{gathered}
p(x_1, 0) = F,\ p(x_1, 1) = T,\ p(x_2, 0) = T, \\
p(x_3, 1) = F,\ p(x_3, 2) = T,\ p(x_4, 1) = F,\ p(x_4, 2) = T \\
p(y_1, 0) = T,\ p(y_2, 0) = T,\ p(y_3, 0) = F,\ p(y_3, 1) = T,\ p(y_4, 0) = T
\end{gathered}$$

These assignments are converted into the following assignments of the 2OPP:

$$x_1 = 1, \quad x_2 = 0, \quad x_3 = 2, \quad x_4 = 2, \quad y_1 = 0, \quad y_2 = 0, \quad y_3 = 1, \quad y_4 = 0$$

These assignments represent the feasible placement shown in Figure 4.1b.

[1]Note that, for Boolean variables $p(x_i, e + w_j)$ and $p(y_i, f + h_j)$, integer e and f satisfy $0 \leq e < W - w_j$ and $0 \leq f < H - h_j$, respectively.

Variables

$p(x_1,0),\ p(x_1,1),\ p(x_1,2),\quad p(y_1,0),\ p(y_1,1),$

$p(x_2,0),\ p(x_2,1),\ p(x_2,2),\quad p(y_2,0),\ p(y_2,1),$

$p(x_3,0),\ p(x_3,1),\qquad\quad p(y_3,0),\ p(y_3,1),\ p(y_3,2),$

$p(x_4,0),\ p(x_4,1),\ p(x_4,2),\quad p(y_4,0),\ p(y_4,1),\ p(y_4,2).$

Axiom Clauses (4.6)

$\neg p(x_1,0) \vee p(x_1,1),\quad \neg p(x_1,1) \vee p(x_1,2),\quad \neg p(y_1,0) \vee p(y_1,1),$

$\vdots$

$\neg p(x_4,0) \vee p(x_4,1),\quad \neg p(x_4,1) \vee p(x_4,2),\quad \neg p(y_4,0) \vee p(y_4,1),\quad \neg p(y_4,1) \vee p(y_4,2).$

Non-overlapping Constraints (4.7), (4.8)

$lr_{1,2} \vee lr_{2,1} \vee ud_{1,2} \vee ud_{2,1},$

$\vdots$

$lr_{3,4} \vee lr_{4,3} \vee ud_{3,4} \vee ud_{4,3}.$

$\neg lr_{1,2} \vee \neg p(x_2,0),\quad \neg lr_{1,2} \vee p(x_1,0) \vee \neg p(x_2,1),$

$\neg lr_{1,2} \vee p(x_1,1) \vee \neg p(x_2,2),\quad \neg lr_{1,2} \vee p(x_1,2),$

$\vdots$

$\neg ud_{3,4} \vee \neg p(y_3,0),\quad \neg ud_{3,4} \vee p(y_4,0) \vee \neg p(y_3,1),$

$\neg ud_{3,4} \vee p(y_4,1) \vee \neg p(y_3,2),\quad \neg ud_{3,4} \vee p(y_4,2).$

Figure 4.2: Example of a SAT-encoded 2OPP

4.4 Solving 2SPP by Incremental SAT Solving

In Section 4.3, we explained how to translate the 2OPP into a SAT problem. In this section, we show how to compute the optimal height of the 2SPP by the incremental SAT solving.

2OPP	Number of rectangles	n
	Integer variables	$2n$
	Constraint clauses	$\frac{n(n-1)}{2}$
SAT-encoded 2OPP	Boolean variables	$O(nW)$
	Axiom clauses	$O(nW)$
	All constraints	$O(n^2W)$

Figure 4.3: Size of the Encoded SAT problem

4.4.1 Searching for Optimum Height of 2SPP

Let *ub* and *lb* be the *upper* and *lower bounds* of a solution to a 2SPP, respectively. In practice, the lower bound is given by exact methods and the upper bound is given by either exact or incomplete methods. Let o be an integer value such that $lb \leq o \leq ub-1$. We introduce a new Boolean variable $p(h, o)$ which is *true* if all rectangles are packed within height o. To solve the 2SPP, for each rectangle r_i, and height o such that $lb \leq o \leq ub - 1$, we have 2-literal clauses:

$$\neg p(h, o) \vee p(y_i, o - h_i) \tag{4.9}$$

Furthermore, for each o ($lb \leq o \leq ub-1$), we have 2-literal axiom clauses due to order encoding:

$$\neg p(h, o) \vee p(h, o + 1) \tag{4.10}$$

Let Ψ be the set of clauses consisting of all clauses obtained from (4.6), (4.7), (4.8), (4.9) and (4.10). Then, we can decide the satisfiability of a 2OPP with the height H by solving the following set of clauses[2]:

$$\Psi \cup \{p(h, H)\} \tag{4.11}$$

[2]That is, we use $p(h, H)$ to restrict the upper bound of the y-coordinate of each rectangle instead of restricting the domain of each rectangle that was defined in Section 4.2.1.

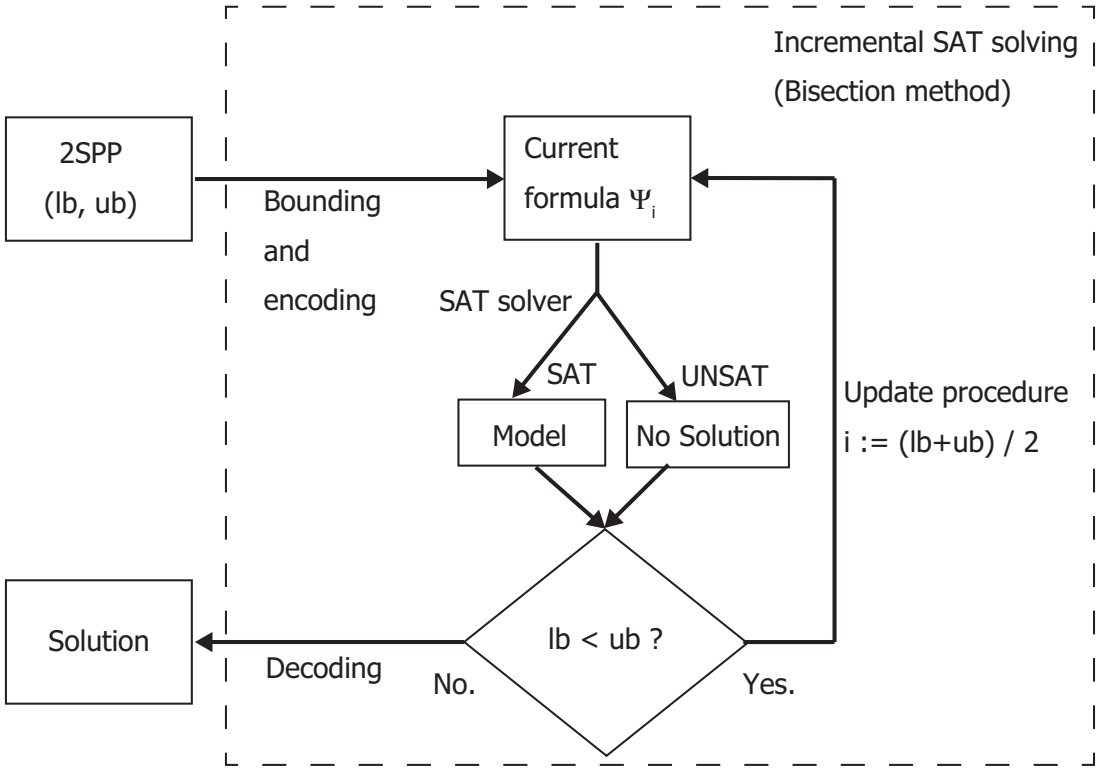

Figure 4.4: Framework for Solving 2OPP

Note that Ψ is common in all 2OPPs associated with the 2SPP. In other words, we bound the 2SPP by adding $p(h, H)$.

Figure 4.4 shows the framework of incremental SAT solving for 2SPP. The optimal height of a 2SPP can be obtained by incrementally solving formulas encoded from 2OPPs. The *bisection method*, which is shown in Section 3.3, is used here. First, we set the lower bound lb and the upper bound ub; then we construct a formula Ψ_{init} of a height which divides the bound region in half. If we find that the encoded 2OPP is satisfiable (`SAT`), the upper bound is updated by the height; otherwise, the lower bound is updated by the height. Then, we compute a new height $i' = \lfloor (lb + ub)/2 \rfloor$. We incrementally perform these operations until we obtain the optimal height, which is on the boundary between the satisfiable and unsatisfiable problems. In this case, the incremental SAT solving is constructed as follows.

(C1) Initial Formula. A propositional formula Ψ_{init} encoded from the 2SPP bounded by the initial bound i.

(C2) Update Procedure. Convert the current formula Ψ_i to the formula $\Psi_{i'}$ that corresponds to the problem bounded by i'.

(C3) Goal Condition. The current bounds do not satisfy $lb < ub$.

For instance, let us consider a 2SPP which has an optimal height of 140. To compute the optimal height, suppose that the lower bound is 48 and that the upper bound is 393. Then the height would change as: 48 (`UNSAT`), 393 (`SAT`), 221 (`SAT`), 134 (`UNSAT`), 178 (`SAT`), 157 (`SAT`), ... The solution area becomes more and more constrained until the optimal height is obtained.

4.4.2 Reusing Learned Clauses

We repeat that when the current assignment leads to a conflict, a new clause indicating the incompatible assignment is generated as a learned clause. For instance, when $(x_1, x_2, x_3) = (T, T, F)$ is the source of a conflict, the clause $\neg(x_1 \wedge x_2 \wedge \neg x_3) = \neg x_1 \vee \neg x_2 \vee x_3$ is generated.

Our SAT encoding approach generates SAT problems incrementally. These SAT problems are similar to each other; that is, Ψ_{o+1} includes all non-unit clauses of Ψ_o. Specifically, in solving the 2SPP, any two 2OPPs in the form (4.11) differ only in their unit clauses $p(h, H)$, which satisfies the reusability condition of learned clauses shown in Proposition 2. We can thus reuse the learned clauses produced while solving the previous 2OPPs in the sequence of encoded 2OPPs.

In addition to reusing learned clauses, we consider to reuse assumptions with the bisection method. In the previous section, we show how to decide the satisfiability at any height of 2SPP, and we need to add a unit clause $\{p(h, H)\}$ to the problem. However, if $\Psi \cup \{p(h, H)\}$ is not satisfied, we cannot continue the bisection method without

removing $\{p(h, H)\}$ from the problem. To resolve this issue, Eén and Sörensson proposed a particular set of unit clauses called *assumptions* [43] and implemented this idea in Minisat. An assumption is added before solving the problem and is then removed from the problem. By adding $p(h, H)$ as an assumption, we can perform the bisection method until the optimal height is obtained.

For instance, let Ψ be an encoded 2SPP with $lb = 4, ub = 10$, and the optimal height is 6. First, we give Ψ and $\{p(h, 7)\}$ as an assumption to the solver, and it returns `SAT`. Now we reuse the assumptions by adding $\{p(h, 7)\}$ to Ψ. Next, we give the SAT solver the problem $\Psi \cup \{p(h, 7)\}$ and an assumption $\{p(h, 5)\}$, and it returns `UNSAT`. Finally, we give the solver the problem $\Psi \cup \{p(h, 7)\} \cup \{\neg p(h, 5)\}$ and an assumption $\{p(h, 6)\}$, and it returns `SAT`. In this way, we can avoid having a redundant search space.

4.4.3 Reduction Techniques

The generated SAT problems grow with the number of input rectangles as shown in Figure 4.3. We thus propose techniques to reduce the search space. We evaluate these techniques in Section 4.5.3.

Large Rectangles. If we are given large rectangles r_i and r_j that satisfy $w_i + w_j > W$, we can assign $lr_{i,j} = false$ and $lr_{j,i} = false$ by using the size relation of the pair of rectangles. We thus can delete all clauses which contain either the literal $\neg lr_{i,j}$ or $\neg lr_{j,i}$ and all other occurrences of the literals $lr_{i,j}, lr_{j,i}$ in every other clause. The condition $w_i + w_j > W$ means we cannot pack rectangles r_i and r_j in the horizontal direction (see Figure 4.5). This reduction technique can also be used in the vertical direction.

Domain reduction. To prune the search space, we reduce the domain of the *largest* rectangle by symmetry breaking (see Figure 4.5a). There are three cases wherein

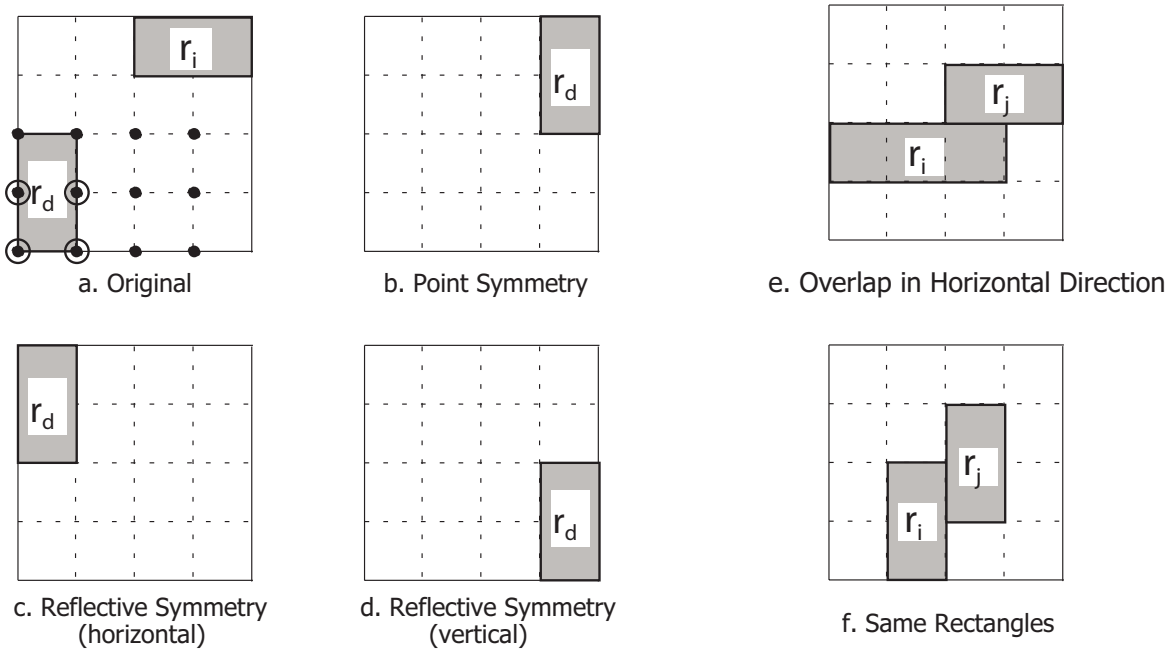

Figure 4.5: Symmetry Breaking and Reduction Techniques

largest can refer to width, height, or area. We choose the rectangles that have the largest width in this chapter. Let (x_d, y_d) be the lower left coordinate of the rectangle r_d and a an integer. Then we can reduce the domain of x_d as follows:

$$D(x_d) = \left\{ a \in \mathbb{N} \;\middle|\; 0 \leq a \leq \left\lfloor \frac{W - w_d}{2} \right\rfloor \right\}$$

Similarly, the domain of y_d can be reduced as follows:

$$D(y_d) = \left\{ a \in \mathbb{N} \;\middle|\; 0 \leq a \leq \left\lfloor \frac{H - h_d}{2} \right\rfloor \right\}$$

In Figure 4.5a, black dots denote the original domain of x_d and y_d, and circles denote the reduced domains described above. The domain $D(x_d)$ is reduced from $\{0, 1, 2, 3\}$ to $\{0, 1\}$.

Applying this reduction, if the rectangle r_i satisfies $w_i > \left\lfloor \frac{W - w_d}{2} \right\rfloor$ then we can assign $lr_{i,d} = false$; that is, we can thus delete all clauses which contain the literal

$lr_{i,d}$ and all other occurrences of the literals $\neg lr_{i,d}$ in every other clause. In Figure 4.5a, we can see that the rectangle r_i, which has $w_i = 2$, cannot be packed to the left of the rectangle r_d because the domain $D(x_d)$ is now $\{0, 1\}$. This reduction is also possible in the vertical direction.

Same Rectangles. If we are given rectangles r_i and r_j which have the same dimensions $(w_i, h_i) = (w_j, h_j)$, we can fix the positional relation of those rectangles (see Figure 4.5f). We consider, for instance, two same rectangles r_i and r_j. If rectangle r_i is placed at the left of r_j in one solution, there must be another solution in which r_j is placed at the left of r_i. We thus assign $lr_{j,i} = false$ and add the clause $lr_{i,j} \vee \neg ud_{j,i}$ to restrict the positional relation of r_i and r_j.

One Pair of Rectangles. We can fix the positional relation between only one pair of rectangles. See Figure 4.5e, by symmetry breaking, we can restrict the positional relation between r_i and r_j. Hereby, we can assign $lr_{j,i} = false$ and $ud_{j,i} = false$. Note that this symmetry breaking cannot simultaneously be used with *domain reduction*.

Exclusive Constraints. Supposing that the rectangle r_i is placed at the left of the rectangle r_j, $lr_{i,j}$ is assigned $true$ and then $lr_{j,i}$ must be assigned $false$. Since $lr_{i,j}$ and $lr_{j,i}$ can never be true simultaneously. We thus can add clauses $\neg lr_{i,j} \vee \neg lr_{j,i}$. Although such a clause may not be necessary, it can accelerate unit propagation during SAT solving.

4.5 Experimental Results

4.5.1 Benchmarks and Environments

We used the following benchmarks to evaluate our method. The set `HT` consists of 9 instances by Hopper and Turton [71]; `BENG` consists of 10 instances by Bengtsson [16];

CGCUT consists of 3 instances by Chiristofides and Whitlock [28]; GCUT consists of 4 instances; and NGCUT consists of 12 instances by Beasley [12, 13]. The total of 38 instances is available at "DEIS - Operations Research Group Library of Instances" [1].

All experiments were executed on a Xeon 2.6GHz. We implemented the program based on Sugar [127] and Minisat [42]. These benchmark sets included some instances which were very hard to solve. In particular, HT08, CGCUT02, CGCUT03, GCUT02 and GCUT04 have not been solved by [97, 3, 138, 4, 105].

4.5.2 Results for 2OPP Instances

Before evaluating proposed method for 2SPPs, we make a comparison with a CSP solver for 2OPP instances. We convert the 2OPP formalized as CSP into the XML format used in the CSP Solver Competition [2].

Table 4.1: Comparison with a CSP Solver [107]

Instance	Height	SAT/UNSAT	CPU Time (sec)	
			cpHydra	Proposal.
HT01(c1p1)	19	UNSAT	—	8.00
	20	SAT	282.61	0.83
HT02(c1p2)	19	UNSAT	—	39.76
	20	SAT	—	2.01
HT03(c1p3)	19	UNSAT	—	20.24
	20	SAT	317.65	0.83
HT04(c2p1)	14	UNSAT	—	—
	15	SAT	—	73.71
HT05(c2p2)	14	UNSAT	—	—
	15	SAT	—	43.76
HT06(c2p3)	14	UNSAT	—	—
	15	SAT	—	1.95

Instance	Height	SAT/UNSAT	CPU Time (sec)	
			cpHydra	Proposal.
HT07(c3p1)	29	UNSAT	—	—
	30	SAT	—	370.62
HT09(c3p3)	29	UNSAT	—	—
	30	SAT	—	369.32
BENG01	29	UNSAT	—	886.95
	30	SAT	—	1.54
BENG02	56	UNSAT	—	—
	57	SAT	—	65.80
BENG03	83	UNSAT	—	—
	84	SAT	—	1246.93
BENG06	35	UNSAT	—	—
	36	SAT	—	3.70
CGCUT01	22	UNSAT	—	443.60
	23	SAT	1.08	0.82
GCUT01	1015	UNSAT	23.38	9.11
	1016	SAT	1.08	1.39
GCUT03	1802	UNSAT	—	—
	1803	SAT	13.70	20.85
NGCUT01	22	UNSAT	—	1.17
	23	SAT	1.08	0.60
NGCUT02	29	UNSAT	—	14.79
	30	SAT	281.70	0.86
NGCUT03	27	UNSAT	—	—
	28	SAT	—	1.41
NGCUT04	19	UNSAT	1.09	0.41

Instance	Height	SAT/UNSAT	CPU Time (sec)	
			cpHydra	Proposal.
	20	SAT	1.09	0.51
NGCUT05	35	UNSAT	—	3.20
	36	SAT	8.37	0.78
NGCUT06	30	UNSAT	—	52.64
	31	SAT	18.78	0.76
NGCUT07	19	UNSAT	1.09	0.29
	20	SAT	1.09	0.51
NGCUT08	32	UNSAT	—	2.87
	33	SAT	383.58	0.75
NGCUT09	49	UNSAT	—	694.51
	50	SAT	—	11.18
NGCUT10	79	UNSAT	—	18.08
	80	SAT	31.73	0.85
NGCUT11	51	UNSAT	—	13.41
	52	SAT	284.16	0.94
NGCUT12	86	UNSAT	—	1.23
	87	SAT	281.28	1.67

Each instance is given one less than the optimal height and the optimal height of the 2SPP in advance e.g., a 2SPP instance NGCUT01 which has the optimal height 23, is converted into two 2OPP instances given height $H = 23$ and $H = 22$, respectively. We choose cpHydra [107] as the CSP solver to be compared. This CSP solver is a portfolio type solver which consists of three solvers, abscon [89], choco [80], and mistral [65], which have different characteristics. The reason why we chose cpHydra is that this solver won the CSP solver competition overall. It came first in four of the five categories.

Table 4.2: Results for 2SPP Instances: HT09, NGCUT09

Instance	Height	SAT/UNSAT	*no-reduce*	*large*	*domain*	*same*	*pair*	*exclusive*
HT09	34	SAT	0.19	0.17	0.16	0.21	0.20	0.16
	32	SAT	0.18	0.18	0.20	0.22	0.25	0.18
	31	SAT	0.25	0.25	0.20	0.34	0.39	0.19
	*30	SAT	595.05	705.15	13.42	719.94	206.74	1265.77
NGCUT09	53	SAT	0.05	0.04	0.08	0.05	0.06	0.05
	51	SAT	0.05	0.07	0.09	0.07	0.07	0.05
	*50	SAT	0.15	6.91	0.25	2.25	1.28	5.95
	49	UNSAT	614.71	675.68	191.82	110.31	489.13	1014.62

In particular, it won "N–ARY–INT" category that included non–binary CSPs such as 2OPP. Table 4.1 shows the results of the comparison on the set of benchmarks reported by Martello *et al.* [97]. All experiments here were executed within 3600 seconds. Note that the problems which cannot be solved by both cpHydra and our method are omitted from the table. Columns 1–3 show the characteristics of each 2OPP instance, such as the instance name, given height, and satisfiability. Column 4 shows the computational time of the CSP solver. "—" indicates time out. Column 5 shows the results of the SAT-based method.

The results listed in the table demonstrate the effectiveness of our SAT-based method. It solved a larger number of instances than cpHydra. cpHydra solved 18 instances, whereas the SAT-based method solved 44 instances.

Table 4.3: Comparison between Combined Methods

Instance	n	W	LB/UB	CPU Time (sec)			Value
				Normal	LCR	LCR+RM	
HT01	16	20	20/25	1.77	1.41	1.40	20
HT02	17	20	20/25	5.90	5.64	3.80	20
HT03	16	20	20/22	1.28	1.22	1.26	20
HT04	25	40	15/17	33.79	101.03	13.88	15
HT05	25	40	15/18	8.11	10.14	23.45	15
HT06	25	40	15/16	4.69	3.24	3.24	15
HT07	28	60	30/35	—	98.23	168.25	30
HT08	29	60	30/35	—	2131.97	—	*30
HT09	28	60	30/37	706.51	252.99	28.81	30
BENG01	20	25	30/37	2.55	2.07	2.44	30
BENG02	40	25	57/64	—	22.97	105.92	57
BENG03	60	25	84/89	—	1309.17	823.77	84
BENG05	100	25	134/142	—	—	3355.16	134
BENG06	40	40	36/38	15.39	10.44	11.15	36
CGCUT01	16	10	23/27	1.44	1.37	1.28	23
GCUT01	10	250	1016/1016	1.58	0.63	1.02	1016
GCUT03	30	250	1803/1803	23.53	4.34	5.98	1803
NGCUT01	10	10	23/25	0.77	0.80	0.74	23
NGCUT02	17	10	30/33	1.60	1.57	1.45	30
NGCUT03	21	10	28/30	1.86	1.90	1.94	28
NGCUT04	7	10	20/20	0.41	0.40	0.41	20
NGCUT05	14	10	36/37	1.03	1.05	0.97	36
NGCUT06	15	10	31/35	1.33	1.36	1.32	31
NGCUT07	8	20	20/20	0.50	0.48	0.52	20
NGCUT08	13	20	33/38	1.18	1.23	0.97	33
NGCUT09	18	20	49/57	823.85	4325.41	4.71	50
NGCUT10	13	30	80/81	1.10	1.05	1.00	80
NGCUT11	15	30	50/57	1.52	1.40	1.25	52
NGCUT12	22	30	87/87	1.90	1.70	1.70	87
Number of Solved Instances				24	28	28	

4.5.3 Results for 2SPP Instances

We evaluated the reduction and reusing techniques described in Sections 4.4.3 and 4.4.2. Table 4.2 shows the results for 2SPP instances with the bisection method. All experiments here were executed within 7200 seconds. We performed the method with each reduction technique on `HT09` and `NGCUT09`. Columns 1–3 show the characteristics of each 2OPP instance, such as the instance name, given height, and satisfiability. "*" indicates that the height is optimal. Columns 4–9 show the CPU time of the reducing techniques: *no-reduce* means the method without reducing techniques, *domain* means symmetry breaking with the largest rectangle, *large* means reduction with large rectangles, *same* means reduction with the same rectangles, *pair* means symmetry breaking with the largest pair of rectangles, *exclusive* means adding exclusive constraints. We can see that there are difficult problems around the boundary between the satisfiable and unsatisfiable problems. We found that the symmetry breaking techniques *domain* and *pair* performed well on `SAT` and `UNSAT` problems. *same* outperformed the other techniques for the NGCUT09 instance because the instance includes five pairs of the same rectangles out of 18 rectangles.

To solve the problems more efficiently, we combined the reduction and reuse techniques. Table 4.3 lists the results. Note that the problems which cannot be solved by all methods are omitted from the table. Columns 1–4 show the characteristics of each instance, such as the instance name, number of input rectangles, strip width W, and the lower bounds from the literature [97]. In this experiment, we used the upper bounds generated by the best-fit algorithm [25]. Although there are more strict lower and upper bounds, we used those for the wider evaluation between our own techniques. In Column 5, `Normal` shows the results of the method without any reducing and reusing. In Column 6, `LCR` shows the results of the method with learned clause reusing. In Column 7, `LCR+RM` shows the results of the method with learned clause reusing and a combination selected from all 24 combinations of reducing methods.

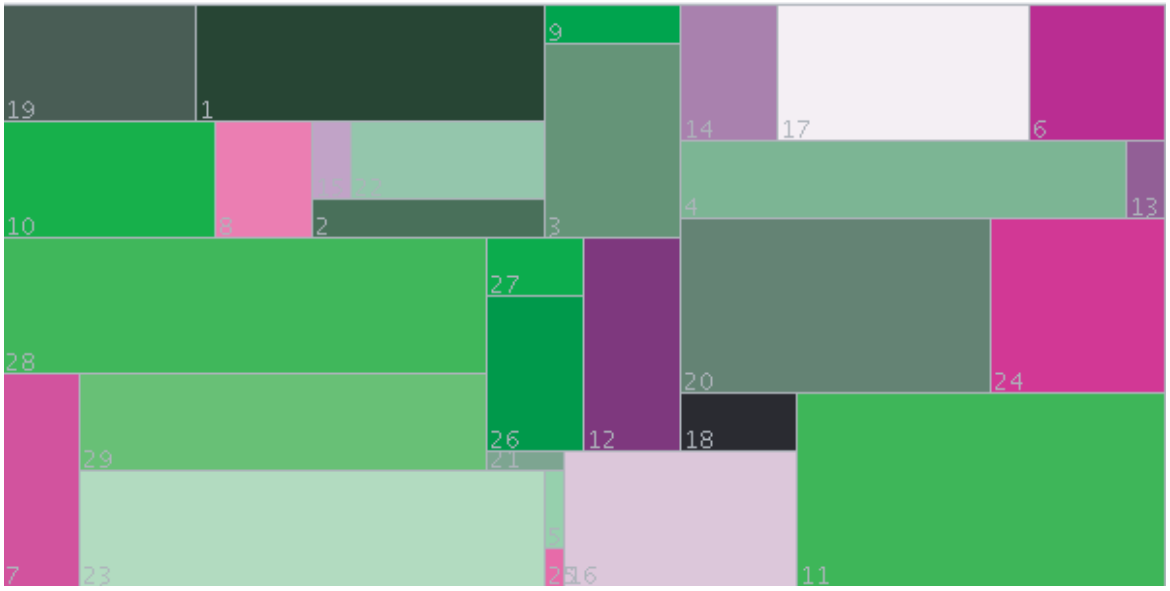

Figure 4.6: Computed Placement of HT08

Specifically, we select the following combination, which solves the largest number of instances with shortest CPU time: reducing with *domain* and *same*.

Column 9 shows the optimal value computed by these three methods. "*" indicates the optimal height that has not been discovered by both exact and incomplete methods [97, 3, 138, 4, 105]. Table 4.3 shows the effectiveness of learned clause reusing. While Normal solved 24 instances, LCR and LCR+RM solved 28 instances. It indicates that the reusing of learned clauses effectively avoid the redundant search space. Furthermore, LCR computes the optimal height of 30 for HT08 (see Figure 4.6) while the best height given by many previous incomplete and exact methods [97, 3, 138, 4, 105] is 31. It also indicates a hidden potential of incremental SAT solving. This is because HT08 cannot be solved by even if we directly compute the 2OPP of its optimal value as we showed it in Figure 4.1. Thus, incremental SAT solving can also be considered as a potential approach for such difficult problems. Besides, reducing techniques are also useful. Although methods of LCR and LCR+RM solved the same number of instances, LCR takes 8297.68 seconds in total while LCR+RM takes 4567.79 seconds. The

Table 4.4: Comparison with the Exact Method by Alvarez-Valdes *et al.* [4]

Instance	n	W	LB/UB [4]	Proposal.		Exact Method [4]	
				Value	Time (sec)	Value	Time (sec)
HT08	29	60	30/31	*30	886.0	31	—
CGCUT02	23	70	63/66	66	—	65	—
CGCUT03	62	70	652/681	681	—	671	—
GCUT02	20	250	1184/1191	1191	—	1190	—
GCUT04	50	250	2995/3003	3003	—	3003	—
NGCUT08	13	20	33/34	*33	0.9	*33	0.1
NGCUT09	18	20	49/50	*50	74.0	*50	48.7
NGCUT11	15	30	51/52	*52	1.2	*52	1.7

method `LCR+RM` includes the reduction *same*, which is particularly effective on `NGCUT` that includes a relatively large number of same-sized rectangles. Thus, `LCR+RM` solved `NGCUT09` quite faster than `LCR`. However, it slows down on `HT08` which is a perfect packing problem and contains only one pair of same rectangles. The combination *same* and *domain* might mislead the search on this instances. Although there is sometimes an exception and we need to adjust combinations, the reduction averagely accelerates searches as is shown by its computation time.

4.5.4 Comparison with Previous 2SPP Method

Recently in 2008, Alvarez-Valdes *et al.* [4] reported a new branch and bound method for 2SPP. They have reported new lower bounds and used an incomplete method called *reactive GRASP* [3] for computing upper bounds. They obtained good bounds for the 2SPP instances described above and found optimal values for 30 out of 38 instances before they applied their new branch and bound method, i.e., their upper

bounds correspond to lower bounds in 30 instances. There are thus only 8 instances remaining to be solved with the branch and bound procedure. Table 4.4 lists results for those 8 instances. Columns 1–4 show the characteristics of each instance, such as the instance name, number of input rectangles, strip width W, and lower bounds from the literature [4]. Columns 5 and 7 show the best value computed by our method and the one by [4]. "*" indicates that the value is optimal. In the case of that the optimal value is computed, we describe CPU time for both methods in columns 6 and 8. Their methods were implemented in C++ and CPLEX, and they were run on a Pentium4 2GHz within 1200 seconds. Our method was also executed within 1200 seconds on the environment shown in Section 4.5.1.

In this experiment, we compared their new branch and bound method with the following method, whose result is the best from all combinations for 8 instances: reducing with *same* and reusing *learned clauses*. We used their lower and upper bounds [4]. Table 4.4 shows that the `NGCUT` instances are easy to solve and both methods solved them within 80 seconds. In contrast, `CGCUT` and `GCUT` are very hard problems, and both methods could not find the optimal solutions. Our method found the optimal value for instance `HT08`, whereas their method could not find a solution. A direct comparison of computational times is not possible because of the difference in machine specs. To give an approximate comparison, their computing times should be divided by 1.3, because we used a Xeon 2.6GHz processor, while Alvarez-Valdes *et al.* used a Pentium4 2GHz. Even if their method could solve `HT08` just after 1200 seconds, its computation time become 923 seconds while our method solved it with 886 seconds.

Besides that, we compare our method with an ad-hoc exact method that employs a branch and bound algorithm by Kenmochi *et al.* [83] for 38 instance described above. It is also difficult to have direct comparison because they use a Pentium4 3GHz processor while our processor is a Xeon 2.6GHz. Table 4.5 shows the result of the number of

Table 4.5: Comparison with the Exact Method by Kenmochi*et al.* [83]

Name	#Instances	LCR	LCR+RM	G-STAIRCASE[83]
HT*	9	9	8	9
BENG*	10	4	5	9
GCUT*	4	2	2	—
CGCUT*	3	1	1	1
NGCUT*	12	12	12	10

solved instances by each method. The columns of `LCR` and `LCR+RM` indicate the results shown in Table 4.3. Nevertheless our processor is slower, `LCR` solved the same number of instances for two benchmark sets: HT* and CGCUT*. Although the proposed methods are inferior in BENG*, both `LCR` and `LCR+RM` overcome in NGCUT*. There is no description about the result of GCUT* in their literature [83].

In this section, we compared our methods with two ad-hoc exact methods. Considering above results, we can say that our proposal is competitive with ad-hoc 2SPP methods for those instance sets.

4.6 Discussion

There are also other well studied encodings that is worth investigating. To consider the difference between order encoding and the others, we compare order encoding with direct encoding, which has been widely used [136]. Let $(w_i, h_i) = (2, 2), (w_j, h_j) = (2, 2)$ and place r_i at $(x_i, y_i) = (3, 3)$ (see Figure 4.7). We can represent non-overlapping constraints between r_i and r_j as follows:

$$(x_j \leq 1) \vee (x_j \geq 5) \vee (y_j \leq 1) \vee (y_j \geq 5)$$

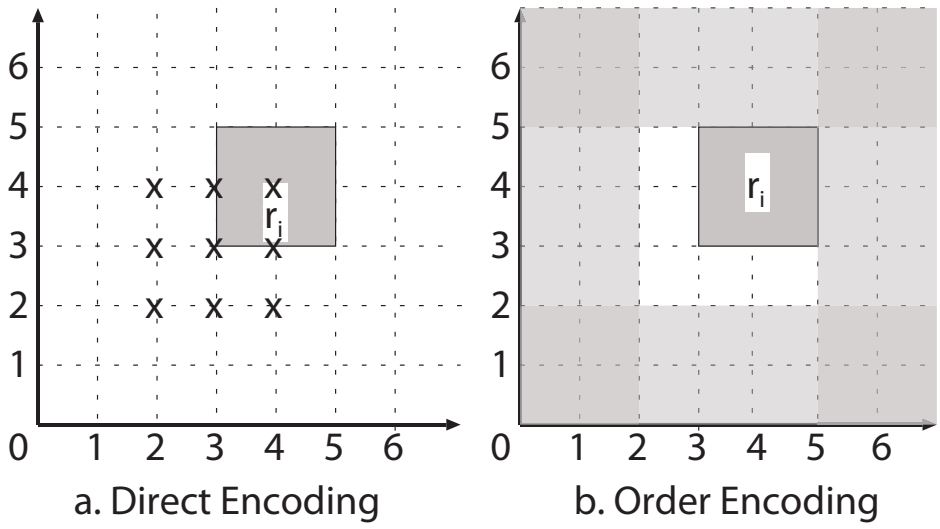

Figure 4.7: Representation of Conflicts

In direct encoding, we assign to a SAT variable $p(x, a) = true$ if and only if the CSP variable x has the domain value a, and the non-overlapping constraints are encoded into the following clauses:

$$
\begin{aligned}
CSP: \quad & (x_j \leq 1) \vee (x_j \geq 5) \vee (y_j \leq 1) \vee (y_j \geq 5) \\
SAT(direct): \quad & \neg p(x_j, 2) \vee \neg p(y_j, 2) \quad \neg p(x_j, 2) \vee \neg p(y_j, 3) \quad \neg p(x_j, 2) \vee \neg p(y_j, 4) \\
& \neg p(x_j, 3) \vee \neg p(y_j, 2) \quad \neg p(x_j, 3) \vee \neg p(y_j, 3) \quad \neg p(x_j, 3) \vee \neg p(y_j, 4) \\
& \neg p(x_j, 4) \vee \neg p(y_j, 2) \quad \neg p(x_j, 4) \vee \neg p(y_j, 3) \quad \neg p(x_j, 4) \vee \neg p(y_j, 4) \\
SAT(order): \quad & p(x_j, 1) \vee \neg p(x_j, 4) \vee p(y_j, 1) \vee \neg p(y_j, 4)
\end{aligned}
$$

Clauses encoded by direct encoding represent conflict points (see Figure 4.7a), and the size of the encoded constraint is $O(n^2W^2)$. In contrast, clauses encoded by order encoding represent these constraints as a single conflict region (see Figure 4.7b), and the size of the encoded constraint is $O(n^2W)$. To encode constraints consisting of comparisons such as $x \leq a$, we can say that order encoding is more suitable than direct encoding. This indicates the SAT-based approach with order encoding would be suitable not only for the 2SPP, but also for other geometric problems such as the

shop scheduling problem.

In this chapter, we use a SAT solver Minisat as a model generator, which is intended to develop for general purpose, and our method on this model generator competes with the latest ad-hoc methods [4, 83]. We think reasons of this good performance are as follows. One is solvers' sophisticated heuristics and optimized implementation that have been accumulated in the last two decades. Another one is a variety of propositional encodings. In the CSP research domain, there are several effective constraint propagation. Those can be emulated by unit propagation in the DPLL procedure through diverse encodings as follows: forward checking; direct encoding [136], maintaining arc consistency; support encoding [55], bounds propagation; order encoding [127, 129]. By choosing appropriate encoding methods, we can do constraint propagation on the established computational engine. Although further analyses are needed, these things can be a reason for the good performance.

4.7 Related Work

There are reports on methods which solve an optimization problem through encoding that translate a problem into a set of decision problems. Bekrar *et al.* [15] reported an approach to the *two-dimensional guillotine strip packing problem* which is a variant of the 2SPP. They represent the problem as CSPs and find the optimum by using the bisection method. The difference from our work is that they directly solved the CSPs whereas we used a SAT-based method. Thereby, we can apply several incremental SAT techniques and use a state-of-the-art SAT solver.

There are several studies on SAT-based methods for other optimization problems. Inoue *et al.* [74] proposed Multisat that can execute several SAT solvers in parallel. They applied it to *SAT Planning* and the *job-shop scheduling problem*. The solvers included in Multisat exchange learned clauses derived by conflict analysis among dif-

ferent SAT solvers. Nabeshima *et al.* [104] reported another approach. Their method shares learned clauses between sub-problems. Our SAT-based approach is an enhancement of these studies; besides reusing learned clauses, we reuse assumptions and newly applied it to the 2SPP. While we applied incremental SAT techniques and symmetry breaking to the 2SPP, Claessen and Sörensson reported a similar method for finding finite models of unsorted first-order logic clause sets [29]. Although there is a difference between our application and theirs, their method includes several incremental SAT techniques and is worth investigating.

4.8 Summary

In this chapter, we apply incremental SAT solving to the two-dimensional strip packing problem. This problem has been mostly investigated from an Operations Research perspective. As far as we know, this is the first attempt that solves the 2SPP with a SAT solver. We compared our method with the state-of-the-art CSP solver, and our method outperformed that solver on 2OPP instances. Besides, we showed that incremental SAT solving is competitive with the previous 2SPP method. In particular, we found the optimal value of `HT08`, which had not been found by almost all of the previous exact and incomplete methods. The result of `HT08` also indicates hidden potential of incremental SAT solving. This is because `HT08` cannot be solved even if we directly compute the 2OPP of its optimal value by a simple SAT solving as is shown in the experiment. Thus, incremental SAT solving can also be considered as a potential approach for such difficult problems. These results indicate that even though SAT-based approaches have been widely studied, there are still challenging topics.

There are several important topics to be tackled. A comparison with other SAT-encoding methods is needed to evaluate the effectiveness of order encoding. Applying order encoding to different problems is important. Enabling rotation of the input

rectangles and the application of other constraints such as *gravity* would be interesting. We could also study other reuse techniques such as the *least number heuristic* [29]. Finally, there is a possibility that a hybrid system composed of incomplete and exact methods can be used to solve larger problems.

Chapter 5

Minimal Active Pathway Finding Problem

In this chapter, we give another application of SAT technologies. We apply incremental SAT solving to a problem in a cross-over research domain, Systems Biology. We propose a new problem to analyze metabolic pathways of organisms. After given the propositional representation of the problem, we explain how to apply incremental SAT solving to it. Following that, we show the experimental results and discuss its meanings from a biological viewpoint.

5.1 Introduction

Living organisms, such as animals, bacteria, fishes and humans, are kept alive by a huge number of chemical reactions. In *Systems Biology*, interactions of such chemical reactions are represented in a network called *pathway*. Analyses of pathways have been active research field in the last decade and several methods have been proposed [79, 131].

A longstanding approach is to represent pathways in a system of differential equa-

tions [133]. Although it is sometimes not easy task to develop it to represent real cell behaviour due to its difficult parameter tuning, this method allows detailed analyses e.g. concentrations of each metabolite with time variation. On the other hand, there are methods aiming for scalable and abstracted analyses have been proposed [33, 115, 110, 120]. This is important because scalability is an important feature for a macroscopical analysis of complex networks like cells. Besides, it is a fundamental goal in Systems Biology. These methods employ discrete approaches rather than continuous ones. The advantage is that it can macroscopically analyze relatively large pathways. These methods are different from each others in their problem formalization and solving methods; however, they share the same purpose, which is to identify biologically functional reaction sets from a given pathway.

One approach relying on graph is proposed by Croes *et al.* [33]. They represent a pathway in a weighted bipartite directed graph and apply a depth-first search algorithm to find the lightest paths from a source compound to a target compound. Planes and Beasley proposed to solve the same problem using a constraint-based method [110]. An advantage of these two methods is that an evaluation of the quality of the solution is provided. We can then choose an objective value to reduce the number of solutions that should be provided to biologists. However, this approach can only generate paths, whereas sub-graphs would be a more natural representation. Moreover, this approach sometimes generates invalid paths from a biological viewpoint because it can easily take non-meaningful shortcuts via common metabolites e.g. water as it indicated by [36].

In this chapter, we propose a new analysis method for metabolic pathways that identifies an *active pathway*, whose form is a sub-graph, which produces a set of target metabolites from a set of source metabolites. In particular, we formalize the problem as finding minimal active pathways, which have the property of not containing any other active pathways. That is, all elements of each minimal active pathway are

qualitatively essential to produce target metabolites. By this property, we can restrict the number of solutions. We represent laws of biochemical multi-molecular reactions in propositional formulas and translate the problem into conjunctive normal form (CNF) formulas. We then use a minimal model generator based on a state-of-the-art SAT solver to solve the problem effectively. Our translation and recent progresses in the SAT domain now make it possible to apply our method to cell-scale pathways. Realistic metabolic pathways include a lot of reversible reactions and cycles. Previous approaches thus needed pre-processing or post-processing, which is possibly costly, to deal with those cycles [110, 128]. We also show how such redundant cycles are avoided in minimal active pathways.

We compare our method with previously proposed approaches [14, 110] for a simplified pathway of *Escherichia coli* (*E. coli*) consisting of 880 reactions. We also test our method with a whole *E. coli* pathway [84] consisting of 1777 reactions. In order to evaluate computed active pathways, we use conventional active pathways described in the literature [14] and EcoCyc [84], which are provided by biological experiments and existing knowledge. As a result, we have identified every conventional active pathway of all pathways we used in the experiments.

In the reminder of this chapter, we explain the minimal active pathway finding problem in Section 5.2. We show the translation from the active pathway finding problem into propositional formulas in Section 5.3. In Section 5.4, we show the experimental result. In Section 5.5 and 5.6 respectively discuss related work and future work.

5.2 Minimal Active Pathway Finding Problem

This section provides the definition of the *minimal active pathway finding problem* (MAPF) on which we are focusing.

Let M a set of metabolites and R a set of reactions. For M and R, $M \cap R = \emptyset$ holds. Let $A \subseteq (R \times M) \cup (M \times R)$ a set of arcs. Let m, r be a metabolite and a reaction such that $m \in M$ and $r \in R$, respectively. A *pathway* is represented in a directed bipartite graph $G = (M, R, A)$ where M and R are two sets of nodes, A is a set of arcs. We here define $M_S \subset M$ as a set of source metabolites and $M_T \subset M$ a set of target metabolites such that $M_S \cap M_T = \emptyset$. A *pathway instance* is represented in a five tuple $\pi = (M, R, A, M_S, M_T)$. A metabolite $m \in M$ is called a *reactant* of a reaction $r \in R$ when there is an arc $(m, r) \in A$. On the other hand, a metabolite $m \in M$ is called a *product* of a reaction $r \in R$ when there is an arc $(r, m) \in A$. A reaction is called a *reversible reaction* if it can occur in both directions between reactants and products. In this research, we distinguish a reversible reaction as two reactions. For instance, if there is a reversible reaction r_1 which has m_1 and m_2 as reactants and m_3 and m_4 as products. In this case, we split the reaction r_1 into two reactions r_{1a} and r_{1b}: one of them has m_1 and m_2 as products and m_3 and m_4 as reactants.

Let $s : R \to 2^M$ be a mapping from a set of reactions to a power set of metabolites such that $s(r) = \{m \in M \mid (m, r) \in A\}$ represents the set of metabolites which are needed to turn the reaction r activatable. Let $p : R \to 2^M$ be a mapping from a set of reactions to a power set of metabolites such that $p(r) = \{m \in M | (r, m) \in A\}$ represents the set of metabolites which are produced by a reaction r. Let $s' : M \to 2^R$ be a mapping from a set of metabolites to a power set of reactions such that $s'(m) = \{r \in R | (m, r) \in A\}$. Let $p' : M \to 2^R$ be a mapping from a set of metabolites to a power set of reactions such that $p'(m) = \{r \in R | (r, m) \in A\}$.

Let t be an integer variable representing time. In this research, an unit of time represent the time of processing any reaction. Besides, we use it to represent order relation of reactions and metabolites. Let $M' \subset M$ be a subset of metabolites. A metabolite $m \in M$ is obviously *producible* at time $t = 0$ from M' on G if $m \in M'$

holds. A reaction $r \in R$ is *activatable* at time $t > 0$ from M' on G if for every $m \in s(r)$, m is producible at time $t-1$ from M'. A metabolite $m \in M$ is *producible* at time $t > 0$ from M' on G if there is at least one activatable reaction r at time t such that $m \in p(r)$.

In this research, we assume that source metabolites are being produced until all target metabolites are generated. This is because common metabolites, e.g. water, can be sufficient amount in a cell and we consider that other source metabolites are enough produced. Thus, reactions using those metabolites can continue to be activatable. Similarly, products of such reactions are continued to be producible. Hence, the following two property hold in pathways. If r is activatable at time t then r is activatable at a time $t+1$. If m is producible at time t then m is producible at time $t+1$.

Definition 9 *Active Pathway*

Let $\pi = (M, R, A, M_S, M_T)$ be a pathway instance and $G = (M, R, A)$ a bipartite directed graph representing a pathway. A sub-graph $G' = (M', R', A')$ of G is an *active pathway* of π if it satisfies the following conditions:

(α1) $M_T \subset M'$

(α2) $M' = M_S \cup \{m \in M \mid (m, r) \subseteq A, r \in R'\} \cup \{m \in M \mid (r, m) \subseteq A, r \in R'\}$

(α3) $A' = \{(m, r) \in A \mid r \in R'\} \cup \{(r, m) \in A \mid r \in R'\}$

(α4) For every $m \in M'$, m is producible from M_S on $G' = (M', R', A')$

From Definition 9, active pathways consist of a set of metabolites and reactions which are producible and activatable from M_S on G' such that all target metabolites M_T are producible. A number of active pathways depend on the combination of M_S and M_T but there is generally a large number of active pathways in a metabolic

pathway. We thus particularly focus on minimal ones rather than active pathways. We give the definition as follows:

Definition 10 *Minimal Active Pathway*

Let $G = (M, R, A)$ and $G' = (M', R', A')$ be active pathways of π, respectively. G is *smaller* than G' and represented in $G \subset G'$ if $R \subset R'$. An active pathway G is *minimal active pathway* of π *iff* there is no active pathway of π which is smaller than G.

From Definition 10, any reactions included in a minimal active pathway cannot be deleted to produce target metabolites. Intuitively, we can understand it as that each of the elements of a minimal active pathway is essential. Finally, the minimal active pathway finding problem is given as follows:

Definition 11 *Minimal Active Pathway Finding Problem (MAPF)*

Input. A pathway instance $\pi = (M, R, A, M_S, M_T)$.

Output. All minimal active pathways of π.

Let z be time to bound the minimal active pathway problem. Specifically, we consider a problem of enumerating all minimal active pathways producing all target metabolites by time z. We call this problem as the minimal active pathway finding problem with regard to z.

Definition 12 *Minimal Active Pathway Finding Problem with regard to z*

Input. A pathway instance $\pi = (M, R, A, M_S, M_T)$ and z.

Output. All minimal active pathways of π by time z.

We can enumerate all minimal active pathways of the original instance by incrementally solving the problems bounded from $z = 1$ to $z = |R|$. In practice, we start to bound the problem to $z = 1$. Then, we compute the resulting minimal active pathways

and analyze those. If we need more solutions, then we increase the bound to $z + 1$ and continue the enumeration. Similar to the above problems, we call the problem of enumerating active pathways as the active pathway finding problem and the active pathway finding problem with respect to z, respectively.

5.3 Incremental SAT Solving for MAPF

In this section, we explain how to apply the *stepwise enumeration* of incremental SAT solving, which is shown in Section 3.4.

5.3.1 Overview

As we describe in the previous section, in most cases, we do not need to enumerate all minimal active pathways of a given pathway instance. In practice, we continue to enumerate solutions until we found a solution in which we are interested. We call this solution as a *preferred solution*. This time, we define the preferred solution in terms of conventional pathways. Specifically, the preferred solution is a pathway included in the conventional pathway of a given pathway instance. Besides, we call the minimal model decoded to a preferred solution as the *preferred minimal model*. To obtain preferred minimal model, we first bound the minimal active pathways by time z. Then, we solve the minimal active pathway finding problem with regard to z and increase the bound z one by one.

(D1) Initial Formula. A propositional formula encoded from MAPF with regard to $z = i$, which is the initial bound.

(D2) Update Procedure. Convert the current formula to the formula that corresponds to MAPF with regard to $i + 1$.

(D3) Goal Condition. The preferred model is found in computed models.

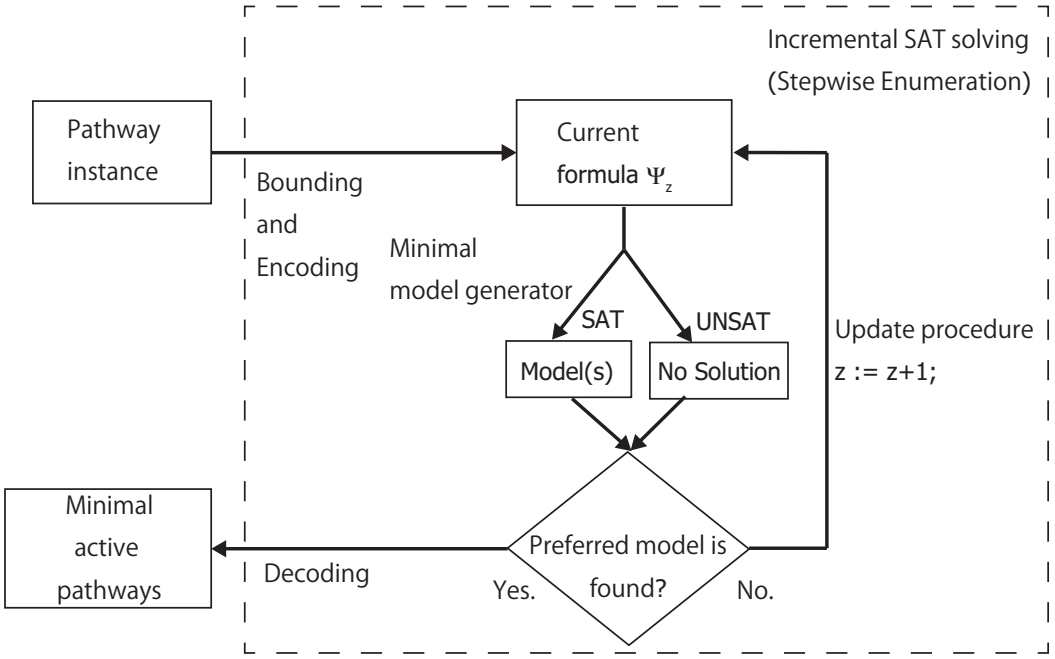

Figure 5.1: The Architecture for Analyzing Metabolic Pathways

The architecture of this approach is shown in Figure 5.1. At first, we are given the minimal active pathway finding problem and bound it to the lowest bound value, which depends each situation but typically it is $z = 1$. Then, the stepwise enumeration of incremental SAT solving is used. We compute all solutions of the MAPF with regard to z and increment the bound z if we cannot find our preferred minimal models. In solving MAPFs, we need a minimal model generator. This time, we use the minimal model generator proposed by Koshimura *et al.* [86] described in Section 2.3.

5.3.2 Reduce Reactions for Encoding

When a pathway instance $\pi = (M, R, A, M_S, M_T)$ is given, we can omit a set of reactions that cannot be activated from a given source metabolites M_S. By deleting such reactions before the encoding, we can reduce the number of clauses. We can find such un-activatable reactions from M_S by the procedure SearchActivatableReactions shown in Figure 5.2. Let Σ be a set of metabolites and Ω a set of reactions that can

SearchActivatableReactions $(G = (M, R, A),\ M_S)$

```
    begin
1:      Σ := M_S;
2:      Ω := {};
3:      while (Σ ≠ {})
4:          mark ∀m_i ∈ Σ as producible;
5:          Σ' = {};
6:          while m_i ∈ Σ do
7:              while r_j ∈ s^{-1}(m_i) do
8:                  if r_j ∉ Ω and ∀m_k ∈ s(r_j) is producible then
9:                      Ω := Ω ∪ {r_j};
10:                 while m_k ∈ p(r_j) do
11:                     if m_k is producible then Σ' := Σ' ∪ {m_k};
12:         Σ := Σ';
13:     return (Ω);
    end
```

Figure 5.2: Procedure for Reduction

be activated from M_S. In the procedure, the first loop begin with the set of source metabolites M_S (Line 1 and 3). Then every metabolites in Σ is marked as producible (Line 4). At Line 8, for every reactions that uses some $m_i \in \Sigma$ as reactants are checked whether activatable or not on current metabolites that are marked as producible. If it is producible then added to Ω that contains activatable reactions. At Line 11, metabolites that can be producible by the reaction r_j are added to Σ' that will be the next seeds of the first loop (Line 11 and 12). Finally, the set of activatable reactions is returned as Σ. By using this, we can reduce the number of encoded clauses.

5.3.3 Encoding

To apply incremental SAT solving to MAPF, we represent it in a propositional formula. An advantage of propositional encoding is that it is flexible for adding biological constraints.

Let i, j be integers denoting indices for metabolites and reactions. Let t an integer variable representing time. Let $\pi = (M, R, A, M_S, M_T)$ a pathway instance. Let z an integer constant representing a given upper bound of the problem. Let V the set of propositional variables which are used in this translation. We introduce two kinds of propositional variables. Let $m^*_{i,t} \in V$ be a propositional variable which is *true* if a metabolite $m_i \in M$ is producible at time t. Let $r^*_{j,t} \in V$ be a propositional variable which is *true* if a reaction $r_j \in R$ is activatable at time t. In the following, we explain the encoding of the minimal active pathway finding problem with regard to z.

For every reaction and metabolite, we have the following formula:

$$\psi_1 = \bigwedge_{0 \leq t < z} \bigwedge_{m_i \in M} \left(m^*_{i,t} \rightarrow m^*_{i,t+1}\right)$$

$$\psi_2 = \bigwedge_{1 \leq t < z} \bigwedge_{r_j \in R} \left(r^*_{j,t} \rightarrow r^*_{j,t+1}\right)$$

These formulas represent that once a metabolite (or a reaction) is changed to producible (or activatable), then it remains in the producible state (or the activatable state).

For each reaction $r_j \in R$, we have the following formula representing that if a reaction r_j is activatable at time t then its reactants must be producible at time $t-1$:

$$\psi_3 = \bigwedge_{1 \leq t \leq z} \bigwedge_{r_j \in R} \left(r^*_{j,t} \rightarrow \bigwedge_{m_i \in s(r_j)} m^*_{i,t-1} \right)$$

For each reaction $r_j \in R$, we have the following formula representing that if a reaction r_j is activatable at time t then its products must be producible at time t.

$$\psi_4 = \bigwedge_{1 \leq t \leq z} \bigwedge_{r_j \in R} \left(r^*_{j,t} \rightarrow \bigwedge_{m_i \in p(r_j)} m^*_{i,t} \right)$$

Table 5.1: Size of Clauses of Ψ_z

Formula	length	#Clauses
ψ_1	2	$\lvert M\rvert * z$
ψ_2	2	$\lvert R\rvert * (z-1)$
ψ_3	2	$\lvert A_{mr}\rvert * z$
ψ_4	2	$\lvert A_{rm}\rvert * z$
ψ_5	more than 3	$\lvert M\rvert * z$
ψ_6	1	$\lvert M\rvert$
ψ_7	1	$\lvert M_T\rvert$

For each metabolite $m_i \in (M \setminus M_S)$, we have the following formula representing that if a reaction m_i is producible then either two states hold: the metabolite m_i is producible at $t-1$ or at least one reaction r_j is activatable.

$$\psi_5 = \bigwedge_{1 \le t \le z} \bigwedge_{m_i \in (M \setminus M_S)} \left(m^*_{i,t} \rightarrow m^*_{i,t-1} \vee \bigvee_{r_j \in p'(m_i)} r^*_{j,t} \right)$$

An initial condition and a target condition are given as follows:

$$\psi_6 = \bigwedge_{m_i \in M_S} m^*_{i,0} \wedge \bigwedge_{m_{i'} \in M \setminus M_S} \neg m^*_{i',0}$$

$$\psi_7 = \bigwedge_{m_i \in M_T} m^*_{i,z}$$

Finally, we have the translated formula Ψ_z as the conjunction of ψ_1, ψ_2, ..., ψ_7. The length of clauses of ψ_6 and ψ_7 is 1, the length of ψ_1, ψ_2, ψ_3 and ψ_4 is 2, ψ_5 is more than 3. Let A_{rm} be a set of arcs such that $A_{rm} = \{(r, m) \in A\}$ and A_{mr} be a set of arcs such that $A_{mr} = \{(m, r) \in A\}$. The size of this encoding is summarized in Table 5.1. Recent SAT solvers are capable to treat over ten millions of clauses; it indicates that the encoding is capable to treat pathways whose number of reactions is over thousands in terms of the size of clauses.

The next proposition gives the soundness and completeness of the encoding for the active pathway finding problem with regard to z.

Proposition 3 Given $\pi = (M, R, A, M_S, M_T)$ and z, let Ψ_z be the translated formula as above. There is a model I of Ψ_z iff $G^{sol} = (M^{sol}, R^{sol}, A^{sol})$ is an active pathway of π producing all target metabolites by time z, where

$M^{sol} = \{m_i \mid m^*_{i,z} \in I\}$,

$R^{sol} = \{r_j \mid r^*_{j,z} \in I\}$, and

$A^{sol} = \{(m_i, r_j) \in A \mid r_j \in R^{sol}\} \cup \{(r_j, m_i) \in A \mid r_j \in R^{sol}\}$.

Proof. ($\Longrightarrow$) Let I be any model of the translated formula Ψ_z. In the following, we prove that G^{sol} satisfies all conditions in Definition 9.

At first, obviously, G^{sol} is a sub-graph of G. Since $\Psi_z \models \psi_7$, G^{sol} satisfies the condition $\alpha 1$. Since $\Psi_z \models \psi_6$, for $m_i \in M_S$, $m^*_{i,z} \in I$ holds. Since $\Psi_z \models \psi_1 \wedge \psi_3 \wedge \psi_4$, if $r^*_{j,z} \in I$ then for any $m_i \in s(r_j) \cup p(r_j)$, $m^*_{i,z} \in I$ holds. Thus G^{sol} satisfies the condition $\alpha 2$. Since the definition of G^{sol}, it satisfies the condition $\alpha 3$. We prove the condition $\alpha 4$ by mathematical induction on the integer variable k such that $0 \leq k \leq z$. (Basis of induction) We prove the case of $k = 0$. Since $\Psi_z \models \psi_1 \wedge \psi_6$, if $m^*_{i,k=0} \in I$ then a metabolite $m_i \in M^{sol}$ is obviously producible from M_S on G^{sol}.
(Induction step) Assume that, if $m^*_{i,k} \in I$ then $m_i \in M^{sol}$ is producible from M_S on G^{sol}. On this induction hypothesis, we prove that if $m^*_{k+1} \in I$ then $m_i \in M^{sol}$ is producible from M_S on G^{sol}. Since $\Psi_z \models \psi_5$, if $m^*_{i,k+1} \in I$ then either the following two cases must hold: (a) $m^*_{i,k} \in I$, (b) for at least one reaction $r_j \in p'(m_i)$, $r^*_{j,k+1} \in I$ holds. In the case of (a), by the induction hypothesis, m_i is producible from M_S at time k. Thus, since the definition of producibility, m_i is producible at time $k + 1$. In the case of (b), since $\Psi_z \models \psi_3$, for any $m_a \in s(r_j)$, $m^*_{a,k} \in I$ holds. Since the induction hypothesis, every metabolite $m_a \in s(r_j)$ is producible at time k. Then, by the definition, r_j is activatable at time $k + 1$. Thus, any metabolites m_i which are product of r_j are producible at time $k + 1$. In both cases (a) and (b), $m_i \in M^{sol}$ is

producible at time $k+1$. Thus, by the mathematical induction, for any k, $m_i \in M^{sol}$ is producible from M_S on G^{sol}. Hence, G^{sol} satisfies the condition $\alpha 4$. Therefore, G^{sol} is an active pathway of π.

($\Longleftarrow$) Let $G^a = (M^a, R^a, A^a)$ be an active pathway of π by time z. We prove that if there exists G^a then there must be a model of Ψ_z that constructs G^a.

Let I_a be a set of propositional variables that constructs G^a. Since G^a is an active pathway, for every $m_i in M^a$, m_i is producible from M_S. Besides, since $M_T \subset M^a$, $I^a \models \psi_5 \wedge \psi_6 \wedge \psi_7$. Moreover, since the conditions $\alpha 2$ and $\alpha 3$, $I^a \models \psi_3 \wedge \psi_4$. For every metabolite $m_i \in M^a$ (or reaction $r_j \in R^a$), since if m_i (or r_j) is producible (or activatable) at time t then the state remains at time $t+1$, $I^a \models \psi_1 \wedge \psi_2$ holds. Hence, $I^a \models \Psi_z$ holds. Therefore, if there exists an active pathway of π then there must be a model of Ψ_z that constructs the active pathway. □

Let V^z be a set of propositional variables of reactions such that $V^z = \{r^*_{i,z} | r_i \in R\}$. The next proposition gives the soundness and completeness of the encoding for the minimal active pathway finding problem with regard to z.

Proposition 4 Given a pathway instance $\pi = (M, R, A, M_S, M_T)$ and z, let Ψ_z be the translated formula as above. I_{min} is a minimal model of Ψ_z with respect to V^z iff $G^{sol} = (M^{sol}, R^{sol}, A^{sol})$ is a minimal active pathway of π producing all target metabolites by time z, where

$M^{sol} = \{m_i \mid m^*_{i,z} \in I_{min}\}$,

$R^{sol} = \{r_j \mid r^*_{j,z} \in I_{min}\}$, and

$A^{sol} = \{(m_j, r_i) \in A \mid r_i \in R^{sol}\} \cup \{(r_i, m_j) \in A \mid r_i \in R^{sol}\}$.

Proof. ($\Longrightarrow$) By the proposition 3, G^{sol} is an active pathway of π. Assume that there exists $G' = (M', R', A')$ such that $G' \subset G^{sol}$. By the definition, $R' \subset R^{sol}$ holds. Then, by the proposition 3, there exists a model I' of Ψ_z, and $I' \cap V^z \subset I_{min} \cap V^z$ holds. However, it contradicts with the definition of the minimal model. Thus, there cannot be such an active pathway. Therefore, G^{sol} is a minimal active pathway.

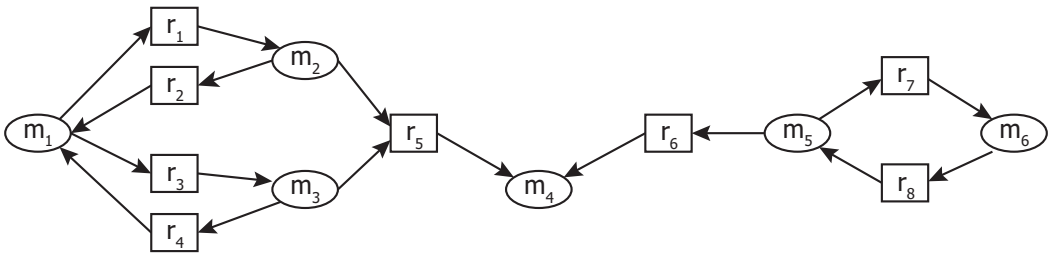

Figure 5.3: Example of Reversible Reactions

($\Longleftarrow$) By the proposition 3, any minimal active pathway has a corresponding model of Ψ_z. Let I_{min} be a model that constructs G^{sol}. Assume that there exists a model I' such that $I' \cap V' \subset I_{min} \cap V'$. Then, an active pathway G' constructed from I' mush satisfy $G' \subset G^{sol}$. However, it contradicts with the definition of the minimal active pathway. Therefore, if there exists a minimal active pathway then there must be a minimal model of Ψ_z with regard to V^z. $\square$

5.3.4 Treating Reversible Reactions and Cycles

Treatment of meaningless reversible reactions and cycles frequently become a problem in pathway analyses because these cause the expansion of the number of useless solutions. For this problem, some previous approaches took pre-processing or post-processing which breaks reversible reactions and meaningless cycles in a pathway [14, 110, 128]. Unlike those approaches, our method resolves the problem by considering the minimality to produce all target metabolites.

For instance, we consider an example including reversible reactions shown in Figure 5.3. Two sets of metabolites $M_S = \{m_1\}$ and $M_T = \{m_4\}$ are given as initial condition. At first, pathways including the reactions r_6, r_7 or r_8 cannot be active pathways because these reactions are not activatable from M_S. Then, suppose that four active pathways G_1, G_2, G_3 and G_4, which consist of $R_1 = \{r_1, r_3, r_5\}$, $R_2 = \{r_1, r_2, r_3, r_5\}$, $R_3 = \{r_1, r_3, r_4, r_5\}$ and $R_4 = \{r_1, r_2, r_3, r_4, r_5\}$, respectively. Obviously, three active pathways G_2, G_3 and G_4 including reversible reactions. However, since there are

smaller active pathways G_1, three active pathways G_2, G_3 and G_4 cannot be minimal active pathways. In this way, we can avoid inclusion of meaningless reversible reactions and cycles in minimal pathways. From minimal pathways, we can see a property that every reaction is essential, that is, none of them can be removed from minimal active pathways to produce target metabolites.

5.3.5 Biological Applications

By the encoding described above, we can obtain minimal active pathways in a normal cell. However, we sometimes want to know minimal pathways in a mutant cell, or there are sometimes other qualitative constraints in actual metabolic pathways. In order to adapt several situations, we can add optional constraints. In the following, we show some of them.

Restricting the Producibility and the Increase of Metabolites

We often encounter the case of that there is a set of metabolites that we do not want to produce in the application such as drug design. In this case, we can add constraints that forbid making such metabolites producible. It can be represented in the following formulas for each time t $(0 \leq t \leq z)$:

$$\bigwedge_{m_i \in M_f} \neg m^*_{i,t}$$

where M^f is a set of metabolites which are forbidden to be producible. Those constraints are useful to refine outputs when we know such forbidden metabolites in advance.

While analyzing organisms, there is a case that producing a metabolite is allowed, but the metabolite must be consumed by some reactions to avoid the increase of the metabolite. In this case, at least one reaction whose reactants include the metabolite must be included in a solution pathway. Let M' be a set of such metabolites. The

constraints represented in the following formulas for each time t ($0 \leq t < z$):

$$\bigwedge_{m_i \in M'} m^*_{i,t} \rightarrow \bigvee_{r_j \in s'(m_i)} r^*_{j,t+1}$$

Constraints related to Gene Regulatory Networks

Besides the producibility of metabolites mentioned above, there is an important constraint related to gene regulation in cells. In general, enzyme, which has an important role of reaction activation, is translated from a gene in the genome. Thus, if there is an exclusive relation between two genes, it means that the two reactions transformed from those genes cannot be activatable, simultaneously. We can add this constraint by the following formulas for each time t ($1 \leq t \leq z$):

$$\neg r^*_{i,t} \vee \neg r^*_{j,t}$$

where reactions r_i and r_j are catalyzed by inhibited enzymes, respectively. This inhibition relation refines output active pathways of the method.

The method allows us to simulate the difference between pathways of wild-type organisms and pathways of mutants or gene knockout organisms. For instance, we can obtain the effect of a gene knock out by removing the reaction r_i related to the gene that we want to delete. This is achieved by adding the following formula for each time t ($1 \leq t \leq z$).

$$\neg r^*_{i,t}$$

As we have shown in this section, not only the standard constraints but also optional constraints can be added. This could make it possible to combine a metabolic pathway with other pathways such as signaling and gene regulatory networks. After adding such constraints, an important property that every reaction is essential is still inherited to solution pathways.

5.4 Experiments and Results

To evaluate the proposed method, we use two reaction databases of *E. coli* K-12. One is from the supplemental data of the literature [14]. Another one is from a well-known biological database *EcoCyc* [84] which gathers results of biological experiments and existence knowledge of *E. coli*. We downloaded the latest version 13.6 of the reaction database of EcoCyc.

In the following experiments, we use conventional active pathways as preferred solutions. Specifically, we said this is a preferred solution when it is included in the conventional pathway. Conventional pathways are respectively obtained from the literature [14] and the database EcoCyc [84]. We modified the Main class of the SAT solver Minisat2 [42] and used it as a minimal model generator by Koshimura *et al.* [86]. In the experiments, before translating a given pathway instance to a propositional formula, we avoid un-activatable reactions from given source metabolites. It is done by a polynomial time procedure based on breadth first search. Its computation time is included in the time of each experiment.

Each experiment has been done using a PC (2.53GHz CPU and 2GB RAM) running Ubuntu Linux 9.04. We have developed a graphical user interface integrating the proposed method, which aims for smooth evaluation. To place the nodes, we use the fast organic layout in the Java library Jgraph. In this layout method, vertexes connected by edges should be drawn close to one another and other vertexes should not be drawn to close to one another. Figures 5.4 and 5.5 are screen shots of our experimental results on the interface.

5.4.1 Comparison with Previous Methods

We compared our method with two previous methods. One is a method using optimization modeling for pathway analyses [14]. An input of this method is a reaction

Table 5.2: Results for Pathways from the Literature [14]

Pathway#	Proposal				[14]		[110]
	z	#S	Time (sec)	cor.	cor. (a)	cor. (b)	cor.
Gluconeogenesis	9	5	1.53	yes	yes	no	no
Glycogen	3	1	0.36	yes	yes	no	yes
Glycolysis	9	35	2.57	yes	yes	yes	no
Proline Bio-synthesis	4	1	0.49	yes	yes	no	no
Ketogluconate Metabolism	5	6	0.76	yes	no	no	yes
Pentose Phosphate	6	7	0.95	yes	yes	no	yes
Deoxythvmidine Phosphate	4	1	0.53	yes	yes	no	yes
Kreb's Cycle	8	35	2.71	yes	no	yes	no
NAD Biosynthesis	5	16	0.69	yes	yes	no	yes
Arginine Biosynthesis	8	1	0.98	yes	yes	no	yes
Total				10	8	2	6

database with stoichiometry. Another one is a constraint based method for path finding [110]. An input of this method is a reaction database without stoichiometry as same as the proposed method. The comparison between these two methods [14, 110] has also shown in the literature [110]. We use same source and target metabolites according to the literature [14]. As right solutions, the method by [110] used liner paths which are chosen from the conventional active pathways of [14]. Similarly, we used those conventional active pathways deleted bypass reactions as right solutions.

The results are shown in Table 5.2. The first column shows 10 pathways we used in this experiment. The second column shows the value of z where the preferred solution was found. The third column shows the number of solutions found by z shown in the second column. The fourth column shows the computation time for each pathway.

The time indicates the total time from $z = 1$ to the value shown in the second column. Columns 5-8 show each result of whether each method could find the pathway or the path corresponding to the conventional one. In columns 6 and 7, (a) represents the objective of minimizing the total number of reactions and (b) represents the objective of maximizing the production of ATP in the literature [14].

As a result, we found every minimal active pathway corresponding to the conventional pathway with time $z \leq 9$. Moreover, the number of solutions is less than 10 except the pathway #3, #8 and #9. By the table, we compare our method with other two methods. However, note that it is difficult to make a direct comparison due to the differences of each input, problem formalization and the number of solutions. While the optimization modeling using stoichiometry information by [14] generates one solution for each pathway, it cannot identify two conventional pathways. Constraint based path finding approach [110] outputs the best 10 paths for each pathway but it cannot identify four conventional pathways. Among three methods, only our method identifies all conventional pathways with tractable number of solutions. The average computation time of the optimization approach is reported that it is 11 seconds over 10 pathways for the two objectives and the longest time is 85 seconds on a machine with 3GHz CPU [14]. There is no mention of computation time in [110]. In contrast, the average computation time of our method over 10 pathways is only 1.2 seconds.

5.4.2 Evaluation on the Whole *E. coli* Metabolic Pathway from EcoCyc

We also apply our method to a whole metabolic pathway of *E. coli*. A bipartite directed graph representation of this pathway is shown in Figure 5.4. In the following, we use the naming of metabolites and reactions used in EcoCyc [84].

We choose source metabolites by calculating percentage of the presence of each metabolites as same as the literature [14]; we define the percentage of the presence

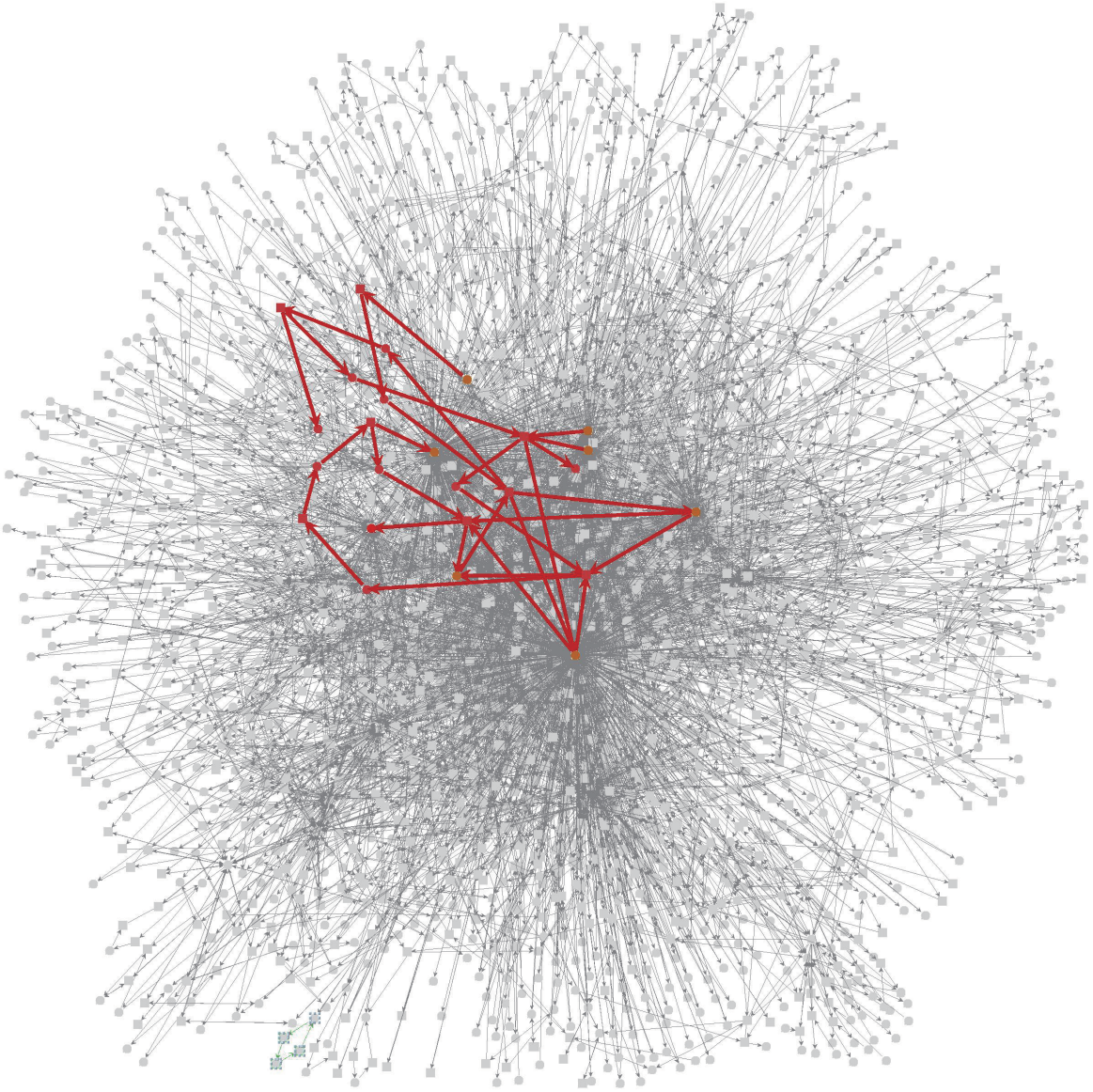

Figure 5.4: Glycolysis Active Pathway on a Whole *E. coli* Pathway

Table 5.3: Computed Minimal Active Pathways

Reaction Name (used in [84])	Reactions of 9 Pathways									
	a	b	c	d	e	f	g	h	i	EC
2.7.1.121-RXN				✓						
2PGADEHYDRAT-RXN				✓	✓	✓	✓	✓	✓	✓
3PGAREARR-RXN				✓	✓	✓	✓	✓	✓	✓
6PFRUCTPHOS-RXN		✓						✓		✓
6PGLUCONOLACT-RXN	✓									
ADENYL-KIN-RXN							✓			
DLACTDEHYDROGNAD-RXN		✓	✓							
F16ALDOLASE-RXN		✓						✓		✓
F16BDEPHOS-RXN										✓
GAPOXNPHOSPHN-RXN				✓	✓	✓	✓	✓	✓	✓
GLU6PDEHYDROG-RXN	✓									
GLYOXIII-RXN		✓	✓							
KDPGALDOL-RXN	✓									
METHGLYSYN-RXN		✓	✓							
NAD-KIN-RXN	✓									
NADPYROPHOSPHAT-RXN						✓				
PEPDEPHOS-RXN					✓			✓		✓
PEPSYNTH-RXN						✓	✓		✓	✓
PGLUCISOM-RXN		✓	✓	✓	✓	✓	✓	✓	✓	✓
PGLUCONDEHYDRAT-RXN	✓									
PHOSGLYPHOS-RXN				✓	✓	✓	✓	✓	✓	✓
RXN0-313			✓	✓	✓	✓	✓		✓	
RXN0-4401									✓	
TRIOSEPISOMERIZATION-RXN			✓							✓
Total Number of Reactions	5	6	6	7	7	8	8	8	8	11

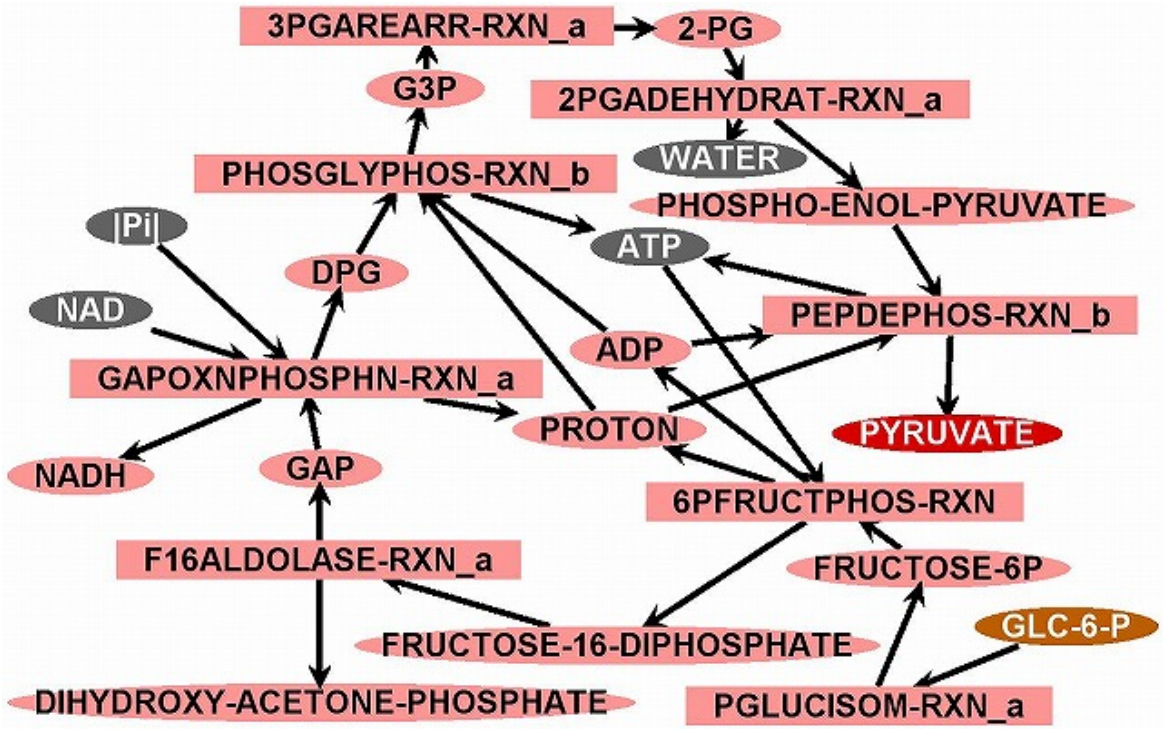

Figure 5.5: Computed Glycolysis Active Pathway of the *E. coli* pathway

of a metabolite m as $pr_m = (n_m \div |R|) \times 100$, where n_m represents the number of reactions in which the metabolite m appears. According to the value of pr_m, the first 6 of 1073 metabolites are chosen: `PROTON`, `WATER`, `ATP`, `ADP`, `|pi|` and `NAD`. In addition, `GLC-6-P` and `PYRUVATE` are given as the source metabolite and the target metabolite, respectively. We then apply our method to find a glycolysis pathway in a whole *E. coli* pathway. As we can see it on biological literature such as [47], glycolysis is known to a pathway constructed by 8 steps. In this experiment, $z = 8$ is thus given, and a number of reactions of pathways is limited to less than 8, which is implemented by constraints of a sequential counter by Sinz [124].

As a result, we found 9 minimal active pathways by incremental SAT solving from $z = 1$ to $z = 8$. The total computation time is only 3.61 second. All 9 minimal active pathways are shown in Table 5.3. In the table, the first column shows reaction names constructing pathways. Columns 2-10 show each found minimal ac-

tive pathway. The 11th column EC shows the conventional glycolysis pathway described in EcoCyc. Among 9 pathways, the computed pathway h corresponds to the conventional pathway. We here consider the computed pathway h shown in Figure 5.5. All reactions included in the computed active pathway are included in the conventional active pathway. However, some reactions included in the conventional glycolysis active pathway are not included in the computed active pathway. This is because conventional pathways in EcoCyc sometimes contain bypass reactions. In the case of the glycolysis active pathway, `TRIOSEPISOMERIZATION-RXN` is such a bypass reaction, which consumes `DIHYDROXY-ACETONE-PHOSPHATE` as a reactant and produces `GAP`. The reason why *E. coli* needs this bypass reaction is that accumulation of `DIHYDROXY-ACETONE-PHOSPHATE` is harmful for its cell [48]. Thus, some organisms including *E. coli* have detoxification pathways of `DIHYDROXY-ACETONE-PHOSPHATE` [47]. That is, this metabolite is needed to be consumed by some reactions when it is produced. As we mentioned in the previous section, if we have such biological knowledge in advance, we can add optional constraints. In this case, we add a constraint representing that if the metabolite `DIHYDROXY-ACETONE-PHOSPHATE` is included in the solution pathway then there must be at least one reaction whose reactants including the metabolite. After adding this constraint, our solution pathway correctly includes this bypass as same as the conventional pathway. However, there are other two reactions in the conventional pathway which are not included in the solution pathway. To analyze why these are needed in glycolysis is a future topic to be considered.

5.5 Related Work

As far as the authors are aware, the exactly same problem of the minimal active pathway finding problem has not yet been formalized. Schuster *et al.* propose a method called elementary mode analyses [120]. They focus on metabolic flux distri-

butions, which are computed by matrix calculus, corresponding to sets of reactions in metabolic pathways. This method can treat multi-molecular reactions while taking into account stoichiometry, and its computational scalability is enough to analyze large pathways. However it tends to generate a large number of solutions without ordering e.g. over 20000 solutions are generated for a pathway including 100 reactions [85]. Even though found solutions are potentially interesting, analyzing all of them through biological experiments would be infeasible task. There are other difference from our method. They use stoichiometry information to solve their problem while our problem only needs the topology of a pathway. Besides, their approach needs to strictly define source metabolites with fixed amount that must be consumed. In contrast, our method treats these as candidates that will be utilized; thus, we can flexibly give source metabolites. Availability of optional constrains is also a merit of the use of our method. Küffer *et al.* report an approach using a Petri net [87]. Although their approach considers producibility and activatability of metabolites and reactions, they do not consider subset minimality of the solution. Croes *et al.* report the path finding problem with weighted graphs. They add a weight for each metabolite node according to its degree. The results are improved compared with the original graph but there is still a remaining problem as it indicated by [36].

Tiwari *et al.* propose a method using a weighted Max-SAT solver [132] to analyze signaling pathways. They translate reaction laws into soft constraint represented in weighted clauses to compute ordered solutions. However, its ordering is sometimes not acceptable from a biological viewpoint since reaction laws that must be held are sometimes violated.

Handorf *et al.* propose the *inverse scope problem* [62]. This is the problem of finding necessary source metabolites from target metabolites. The two differences between their problem and our proposal are as follows. One is that they only calculate the cardinality minimal solution. While their approach, we can generate subset minimal

solution with a SAT solver based minimal model generator. Another one is that each of their solution includes all reactions which are activatable from source metabolites which are needed to generate target metabolites. For instance, if there are two ways to produce a metabolite from source metabolites then both are mixed in one solution, that is, we cannot distinguish the two ways. Our method can distinguish such two ways and we think that it is important to detect functionally minimal pathways for the analyses of metabolic pathways. Schaub and Thiele apply answer set programming (ASP) to the inverse scope problem [119]. There is another research using ASP. Ray *et al.* report a method using ASP to compute the steady states of a given pathway and complete lacking reactions [116]. Unlike their approach, we use minimal model generation to compute essential reactions to produce target metabolites.

5.6 Summary

In this chapter, we formalized the minimal active pathway finding problem that detects minimal reaction sets to produce target metabolites. We then apply the incremental SAT solving to the problem and have an experimental evaluation. Our method based on incremental SAT solving has the following features. It can treat reversible reactions and cycles without pre-processing and post-processing. Besides, it is capable to treat a whole *E. coli* metabolic pathway and optional constraints are additional without modification of the computing procedure. As far as the authors know, there are still few methods have been reported for analyses of a whole organism pathway. Our method provides a new analysis method for a cell-scale metabolic pathway and could treat extended pathways combined with other pathways such as signaling and gene regulatory networks by additional constraints over pathways. As a future topic, considering ranking method for obtained solutions and translating more biological knowledge are important for the analyses of the extended pathways.

Chapter 6

Conclusions and Future Work

This thesis investigates the studies on applying incremental SAT solving to optimization and enumeration problems. So far, SAT problems and solvers have been paid much attention in computer science. Due to its progress, methods based on these techniques are now capable to tackle to a wide range of real world problems. This chapter draws the conclusions from the previous chapters and discusses several future topics.

6.1 Incremental SAT Solving and Applications

Following the consecutive papers by Davis *et al.* in 60's, a large number of researches have been done for both theoretical and practical aspects of SAT. Since around 2000, sophisticated implementations have been reported besides improving algorithms. Several SAT solver competitions push up its practical potential. This thesis particularly studies how to apply SAT techniques to optimization and enumeration problems, which is explained as incremental SAT solving, and apply it to real world problems. So far, some researchers report incremental methods for their specific problems. We construct the incremental SAT solving method consisting of an initial formula, an update procedure and a goal condition. This is so general definition that it could be

applied to many problems. In this solving method, we can utilize learned clauses to effectively compute a sequence of formulas.

We then show that how to solve a variety of problems by the incremental SAT solving and evaluate it in the following two problems. One is the two-dimensional strip packing problem (2SPP). This problem has been actively investigated from an Operations Research perspective. As a new approach for 2SPP, we apply incremental SAT solving. As far as we are aware, this is the first research to solve the 2SPP by SAT technologies. Some of the obtained results are as follows: we compared our method with the state-of-the-art CSP solver, and our method outperformed it on 2OPP instances; we found learned clause reusing and reducing techniques are effective for solving 2SPP instances; we showed that our method is competitive with the previous ad-hoc 2SPP methods. In particular, the result of `HT08` indicates hidden potential of incremental SAT solving. This is because `HT08` cannot be solved even if we directly compute the 2OPP of its optimal value by a simple SAT solving as is shown in the experiment. Thus, incremental SAT solving can also be considered as a potential approach for such difficult problems. These results indicate that even though SAT-based approaches have been widely studied, there are still challenging topics.

In addition, we investigate to apply incremental SAT solving to Systems Biology, which has not been paid so much attention as application of SAT technologies. To analyze metabolic pathways, we propose the minimal active pathway finding problem. This is the problem of identifying necessary reactions to produce target metabolites. Besides, we presented a translation from the problem into a propositional formula. We use the incremental SAT solving to the problem and it has the following features: it can treat reversible reactions without pre-processing and post-processing; it is capable to treat a whole *E. coli* metabolic pathway and optional constraints are additional without modification of the computing procedure. As far as we are aware, there are few methods have been reported for the analysis of a whole organism pathway. Besides

its computational performance, we found 9 minimal active pathways and one of them corresponds to the conventional glycolysis metabolic pathway of *E. coli*. Moreover, we explain that the capability of additional constraints and show it improves solution pathways. These results indicate that the minimal active pathway finding problem is a potential approach for the analysis of metabolic pathways.

6.2 Future Work

6.2.1 Encoding Methods and Other Representation

This thesis addressed the incremental SAT solving and applications to real world problems. In this process, we first need to encode problems in propositional formulas; however, propositional encoding is not unique even for a same problem [76, 136, 55, 52, 127, 129, 130]. Since computational performance of SAT solving depends on encoding, getting more understanding of the relation between encoding and problem structure and finding better encoding are necessary. Both must push up the potential of SAT-based applications. On the other hand, except propositional formulas, there are several ways to represent problems. For instance, many problems can be represented in other frameworks such as constraint satisfaction problem (CSP), answer set programming (ASP). For each representation, there is its own method to solve the problem with its own representation. It is still open problem that which approach is the best. There are some researches that analyze the difference between two approaches [45, 17, 40, 22, 54]. However, more practical and theoretical analyses are necessary. Besides representation, there is another way to solve the problem represented in CSP and ASP, with SAT solvers using some encoding method; Actually, several researches have been done to solve CSP and ASP with SAT solvers [127, 92, 54]. Under this circumstance, we can say that propositional representation e.g. CNF now becomes a kind of assemble language because it is more simple and tractable for computers.

More integrated analysis over several representations is an interesting future topic.

6.2.2 Accelerating Incremental SAT Solving

Besides representation, accelerate solving process for multiple formulas should be considered. For this issue, of course, the development of SAT solvers, is one direction. Another one would be better management of learned clause reusing and variable selection for the incremental SAT solving. Different to single SAT solving, we can measure another statistics while solving previous problems. To utilize such information is a potential direction. In the CSP research domain, a various techniques have been proposed to solve a sequence of CSPs [135]. Transporting these techniques to the incremental SAT solving is important. As we have seen in this thesis, incremental SAT solving can be applied to a wide range of combinatorial problems including optimization and enumeration problems. As a general and powerful approach, it is a promising method and we should have more study on this issue.

6.2.3 Further Applications and Extensions

This thesis describes an application to 2SPP solving. Around this, other important future work is to extend SAT-based methods to other optimization problems and having a comparison with ad-hoc methods. There are many challenging problems that have not been solved by SAT-based methods. Accumulating these comparisons reveals strong and weak points of the methods, and it would promote the better understanding of the advantage of the SAT-based methods. The other application we focused in this thesis is the minimal active pathway finding problem. Since it is difficult to uniquely define which solution is the best one, we need to enumerate solutions in this kind of problems. Although all obtained solutions are possible candidates for the problem, we want to order those in preferred ordering. There is a preceded research on this topic [73]. To adapt such solution ranking method is an interesting topic. In recent

years, symbolic approaches to Systems Biology are rising up. Since propositional formulas are suitable for symbolic representation, it is a good application of SAT solving methods. From a biological perspective, extending the notion of minimal active pathways to other biological network such as gene regulatory networks and signaling networks is important. We believe that this method provides a new approach to whole cell analyses.

Bibliography

[1] DEIS — Operations Research group library of instances. `http://www.or.deis.unibo.it/research_pages/ORinstances.htm`.

[2] Third international CSP solver competition. `http://cpai.ucc.ie/08/`.

[3] R. Alvarez-Valdes, F. Parreno, and J. M. Tamarit. Reactive GRASP for the strip-packing problem. *Computers and Operations Research*, 35(4):1065–1083, 2008.

[4] R. Alvarez-Valdes, F. Parreno, and J. M. Tamarit. A branch and bound algorithm for the strip packing problem. *OR Spectrum*, 31(2):431–459, 2009.

[5] G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of IJCAI-2009*, pages 399–404, 2009.

[6] F. Bacchus, S. Dalmao, and T. Pitassi. Algorithms and complexity results for #SAT and bayesian inference. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03)*, pages 340–351, Washington, DC, USA, 2003.

[7] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In *Proceedings of the Principles and Practice of Constraint Programming (CP'03)*, pages 108–122, 2003.

[8] O. Bailleux and Y. Boufkhad. Full CNF encoding: The counting constraints case. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, 2004.

[9] O. Bailleux, Y. Boufkhad, and O. Roussel. A translation of pseudo Boolean constraints to SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:191–200, 2006.

[10] B. S. Baker, E. G. Coffman Jr., and R. L. Rivest. Orthogonal packings in two dimensions. *SIAM Journal of Computing*, 9(4):846–855, 1980.

[11] R. J. Bayardo and R. C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of AAAI-97*, pages 203–208, 1997.

[12] J. E. Beasley. Algorithms for unconstrained two-dimensional guillotine cutting. *The Journal of the Operational Research Society*, 36(4):297–306, 1985.

[13] J. E. Beasley. An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64, 1985.

[14] J. E. Beasley and F. J. Planes. Recovering metabolic pathways via optimization. *Bioinformatics*, 23(1):92–98, 2007.

[15] A. Bekrar, I. Kacem, C. Chu, and C. Sadfi. A dichotomical algorithm for solving the 2D guillotine strip packing problem. In *Proceedings of the 37th International Conference on Computers and Industrial Engineering*, pages 1216–1224, 2007.

[16] B. Bengtsson. Packing rectangular pieces – A heuristic approach. *Computer Journal*, 25(3):353–357, 1982.

[17] H. Bennaceur. A comparison between SAT and CSP techniques. *Constraints*, 9(2):123–138, 2004.

[18] H. Bennaceur, I. Gouachi, and G. Plateau. An incremental branch-and-bound method for the satisfiability problem. *INFORMS Journal on Computing*, 10(3):301–308, March 1998.

[19] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic model checking using sat procedures instead of bdds. In *Proceedings of the 36th Conference on Design Automation*, pages 317–320, 1999.

[20] A. Biere, M. Heule, H. v. Maaren, and T. Walsh. *Handbook of Satisfiability*. IOS Press, 2009.

[21] M. Bohm and E. Speckenmeyer. A fast parallel SAT-solver – efficient workload balancing. *Annals of Mathematics and Artificial Intelligence*, 17(2):381–400, 1996.

[22] L. Bordeaux, Y. Hamadi, and L. Zhang. Propositional satisfiability and constraint programming: A comparative survey. *ACM Computing Surveys*, 38(4), 2006.

[23] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[24] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[25] E. K. Burke, G. Kendall, and G. Whitwell. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research*, 52(4):655–671, 2004.

[26] C.-L. Chang and R. C.-T. Lee, editors. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.

[27] C. Chaouiya. Petri net modelling of biological networks. *Briefings in Bioinformatics*, 8(4):210–219, 2007.

[28] N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25(1):30–44, 1977.

[29] K. Claessen and N. Sörensson. New techniques that improve mace-style finite model finding. In *Proceedings of the CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.

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

[31] G. L. Cravo, G. M. Ribeiro, and L. A. N. Lorena. A greedy randomized adaptive search procedure for the point-feature cartographic label placement. *Computers and Geosciences*, 34(4):373–386, 2008.

[32] J. M. Crawford and A. B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 1092–1097, 1994.

[33] D. Croes, F. Couche, S. J. Wodak, and J. v. Helden. Inferring meaningful pathways in weighted metabolic networks. *Journal of Molecular Biology*, 356(1):222–236, 2006.

[34] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[35] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[36] L. F. d. Figueiredo, S. Schuster, C. Kaleta, and D. A. Fell. Can sugars be produced from fatty acids?; a test case for pathway analysis tools. *Bioinformatics*, 24(22):2615–2621, 2008.

[37] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pages 37–42, 1988.

[38] G. Dequen and O. Dubois. An efficient approach to solving random k-sat problems. *Journal of Automated Reasoning*, 37(4):261–276, 2006.

[39] J. Desel and G. Juhas. what is a Petri net? informal answers for the informed reader. In H. Ehrig, J. Padberg, G. Juhas, and G. Rozenberg, editors, *Unifying Petri Nets*, volume 2128 of *LNCS*, pages 1–25. Springer, 2001.

[40] Y. Dimopoulos and K. Stergiou. Propagation in CSP and SAT. In *Proceedings of the Principles and Practice of Constraint Programming (CP'06)*, volume 4204 of *LNCS*, pages 137–151. Springer, 2006.

[41] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, January 1990.

[42] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, pages 502–518, 2003.

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

[44] N. Eén and N. Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[45] W. Faber, N. Leone, and G. Pfeifer. A comparison of heuristics for answer set programming. In *Proceedings of the 5th Dutch-German Workshop on Nonmonotonic Reasoning Techniques and their Applications (DGNMR 2001*, pages 64–75, 2001.

[46] B. Faltings and S. Macho-Gonzalez. Open constraint programming. *Artificial Intelligence*, 161(1-2):181–208, 2005.

[47] G. P. Ferguson, S. Totemeyer, M. J. MacLean, and I. R. Booth. Methylglyoxal production in bacteria: suicide or survival? *Archives of Microbiology*, 170(4):209–218, 1998.

[48] W. B. Freedberg, W. S. Kistler, and E. C. C. Lin. Lethal synthesis of methylglyoxal by escherichia coli during unregulated glycerol metabolism. *Journal of Bacteriology*, 108(1):137–144, 1971.

[49] J. W. Freeman. *Improvements to propositional satisfiability search algorithms*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 1995. UMI Order No. GAX95-32175.

[50] B. N. Freeman-Benson, J. Maloney, and A. Borning. An incremental constraint solver. *Communications of the ACM*, 33(1):54–63, 1990.

[51] M. R. Garey and D. S. Johnson, editors. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. FREEMAN AND COMPANY, NY, USA, 1979.

[52] M. Gavanelli. The log-support encoding of CSP into SAT. In *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming (CP'07)*, volume 4741 of *LNCS*, pages 815–822. Springer, 2007.

[53] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *In: Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 386–392, 2007.

[54] M. Gebser and T. Schaub. Characterizing ASP inferences by unit propagation. In *Proceedings of the International Workshop on Logic and Search (LaSh 2006)*, pages 41–56, 2006.

[55] I. P. Gent. Arc consistency in SAT. In *Proceedings of the 15th Eureopean Conference on Artificial Intelligence (ECAI 2002)*, pages 121–125, 2002.

[56] V. Gogate and R. Dechter. Approximate counting by sampling the backtrack-free search space. In *Proceedings of the 22nd National Conference on Artificial Intelligence (AAAI 2007)*, pages 198–203, 2007.

[57] E. Goldberg, M. Prasad, and R. Brayton. Using sat for combinational equivalence checking. In *Proceedings of the conference on Design, automation and test in Europe (DATE '01)*, pages 114–121, 2001.

[58] C. Gomes, H. Kautz, A. Sabharwal, and B. Selman. *Handbook of Knowledge Representation*, chapter Satisfiability Solvers. Elsevier, 2008.

[59] C. Gomes, B. Selman, and N. Crato. Heavy-tailed distributions in combinatorial search. In *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP '97)*, pages 121–135, 1997.

[60] C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the 15th National/10th Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 431–437, 1998.

[61] S. Gopalakrishnan, V. Durairaj, and P. Kalla. Integrating CNF and BDD based SAT solvers. In *Proceedings of the IEEE International High-level Design Validation and Test Workshop (HLDVT 03)*, pages 51–56, 2003.

[62] T. Handorf, N. Christian, O. Ebenhöh, and D. Kahn. An environmental perspective on metabolism. *Journal of Theoretical Biology*, 252(3):530 – 537, 2008.

[63] R. Hasegawa, H. Fujita, and M. Koshimura. MGTP: A model generation theorem prover: Its advanced features and applications. In D. Galmiche, editor, *Automated Reasoning with Analytic Tableaux and Related Methods*, volume 1227 of *LNCS*, pages 1–15. Springer, 1997.

[64] R. Hasegawa, M. Koshimura, and H. Fujita. MGTP: A parallel theorem prover based on lazy model generation. In D. Kapur, editor, *Proceedings of the 11th Conference on Automated Deduction (CADE-11)*, volume 607 of *LNCS*, pages 776–780. Springer, 1992.

[65] E. Hebrard. Mistral, a constraint satisfaction library. In *Proceedings of the Third International CSP Solver Competition*, pages 31–39, 2008.

[66] F. Heras, J. Larrosa, and A. Oliveras. Minimaxsat: An efficient weighted max-sat solver. *Journal of Artificial Intelligence Research*, 31:1–32, 2008.

[67] M. Heule. *SmArT solving: Tools and techniques for satisfiability solvers*. PhD thesis, Delft University of Technology, Delft, The Netherlands, 2008.

[68] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *Journal of the ACM*, 32(1):130–136, 1985.

[69] R. Hofestädt. A Petri net application to model metabolic processes. *Systems Analysis Modelling Simulation*, 16(2):113–122, October 1994.

[70] J. N. Hooker. Solving the incremental satisfiability problem. *Journal of Logic Programming*, 15(1-2):177–186, 1993.

[71] E. Hopper and R. Turton. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research*, 128(1):34–57, 2000.

[72] J. Huang. The effect of restarts on the efficiency of clause learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2318–2323, 2007.

[73] K. Inoue, T. Sato, M. Ishihata, Y. Kameya, and H. Nabeshima. Evaluating abductive hypotheses using an em algorithm on bdds. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 810–815, 2009.

[74] K. Inoue, T. Soh, S. Ueda, Y. Sasaura, M. Banbara, and N. Tamura. A competitive and cooperative approach to propositional satisfiability. *Discrete Applied Mathematics*, 154(16):2291–2306, 2006.

[75] K. Inoue and N. Tamura. Foundations of sat solvers. *Jounal of the Japanese Society for Artificial Intelligence*, 25(1):57–67, 2010.

[76] K. Iwama and S. Miyazaki. SAT-variable complexity of hard combinatorial problems. In *Proceedings of the World Computer Congress of the IFIP*, pages 253–258, 1994.

[77] R. G. Jeroslow and J. Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[78] H. Jin and F. Somenzi. An incremental algorithm to check satisfiability for bounded model checking. *Electronic Notes in Theoretical Computer Science*, 119(2):51–65, 2004.

[79] H. D. Jong. Modeling and simulation of genetic regulatory systems: A literature review. *Journal of Computational Biology*, 9:67–103, 2002.

[80] N. Jussien, C. Prud'homme, H. Cambazard, G. Rochart, and F. Laburthe. choco: an open source Java constraint programming library. In *Proceedings of the Third International CSP Solver Competition*, pages 7–13, 2008.

[81] H. Kautz and B. Selman. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI '92)*, pages 359–363, 1992.

[82] H. A. Kautz, D. Mcallester, and B. Selman. Encoding plans in propositional logic. In *Proceedings of the 5th International Conference on the Principle of Knowledge Representation and Reasoning (KR'96)*, pages 374–384, 1996.

[83] M. Kenmochi, T. Imamichi, K. Nonobe, M. Yagiura, and H. Nagamochi. Exact algorithms for the two-dimensional strip packing problem with and without rotations. *European Journal of Operational Research*, 198(1):73–83, 2009.

[84] I. Keseler, C. Bonavides-Martinez, J. Collado-Vides, S. Gama-Castro, R. Gunsalus, D. Johnson, M. Krummenacker, L. Nolan, S. Paley, I. Paulsen, M. Peralta-Gil, A. Santos-Zavaleta, A. Shearer, and P. Karp. Ecocyc: A comprehensive view of escherichia coli biology. *Nucleic Acids Research*, 37:D464–D470, 2009.

[85] S. Klamt and J. Stelling. Combinatorial complexity of pathway analysis in metabolic networks. *Molecular Biology Reports*, 29(1-2):233–236, 2002.

[86] M. Koshimura, H. Nabeshima, H. Fujita, and R. Hasegawa. Minimal model generation with respect to an atom set. In *Proceedings of the the 7th International Workshop on First-Order Theorem Proving (FTP '09)*, pages 49–59, 2009.

[87] R. Küffner, R. Zimmer, and T. Lengauer. Pathway analysis in metabolic databases via differetial metabolic display (DMD). In *German Conference on Bioinformatics*, pages 141–147, 1999.

[88] E. Lamma, P. Mello, M. Milano, R. Cucchiara, M. Gavanelli, and M. Piccardi. Constraint propagation and value acquisition: why we should do it interactively. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 468–477, 1999.

[89] C. Lecoutre and S. Tabary. Abscon 112 toward more robustness. In *Proceedings of the Third International CSP Solver Competition*, pages 41–47, 2008.

[90] C. M. Li, U. Picardie, J. Verne, and F. Many. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:2007, 2007.

[91] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 366–371. Morgan Kaufmann, 1997.

[92] F. Lin and Y. Zhao. Assat: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1–2):115–137, 2004.

[93] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[94] J. P. Marques-Silva and K. A. Sakallah. GRASP—A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[95] J. Marques-Silva, M. Janota, and I. Lynce. On computing backbones of propositional theories. In *Proceeding of the 19th European Conference on Artificial Intelligence (ECAI 2010)*, pages 15–20, 2010.

[96] K. Marriott, P. Stuckey, V. Tam, and W. He. Removing node overlapping in graph layout using constrained optimization. *Constraints*, 8(2):143–171, April 2003.

[97] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15(3):310–319, 2003.

[98] H. Matsuno, C. Li, and S. Miyano. Petri net based descriptions for systematic understanding of biological pathways. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E89-A(11):3166–3174, November 2006.

[99] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV 2002)*, pages 250–264, 2002.

[100] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)*, pages 25–32, 1990.

[101] A. Morgado and J. Marques-Silva. Good learning and implicit model enumeration. In *Proceeding of the 17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, pages 131–136, 2005.

[102] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of the 38th Conference on Design Automation (DAC '01)*, pages 530–535, 2001.

[103] H. Nabeshima and T. Soh. Principles of modern sat solvers. *Jounal of the Japanese Society for Artificial Intelligence*, 25(1):68–76, 2010.

[104] H. Nabeshima, T. Soh, K. Inoue, and K. Iwanuma. Lemma reusing for SAT based planning and scheduling. In *Proceedings of the International Conference on Automated Planning and Scheduling 2006 (ICAPS'06)*, pages 103–112, 2006.

[105] B. Neveu and G. Trombettoni. Strip packing based on local search and a randomized Best-fit. In *Proceedings of Workshop on Bin Packing and Placement Constraints (BPPC'08)*. 9 pages, 2008.

[106] I. Niemelä. A tableau calculus for minimal model reasoning. In *Proceedings of the Theorem Proving with Analytic Tableaux and Related Methods (TABLEAUX 1996)*, volume 1071 of *LNCS*, pages 278–294, 1996.

[107] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan. Using case-based reasoning in an algorithm portfolio for constraint solving. In *Proceedings of the 19th Conference on Artificial Intelligence and Cognitive Science*, 2008.

[108] C. A. Petri. *Communication with Automata (in German)*. PhD thesis, Schriften des Instituts ür Instrumentelle Mathematik, Bonn, 1962. UMI Order No. GAX95-32175.

[109] K. Pipatsrisawat and A. Darwiche. RSat 2.0: SAT solver description. Technical Report D–153, Automated Reasoning Group, Computer Science Department, UCLA, 2007.

[110] F. J. Planes and J. E. Beasley. Path finding approaches and metabolic pathways. *Discrete Applied Mathematics*, 157(10):2244–2256, 2009.

[111] S. Prestwich. *Handbook of Satisfiability*, chapter 12: CNF encodings, pages 75–97. IOS Press, 2009.

[112] S. D. Prestwich. Local search on SAT-encoded colouring problems. In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 105–119, 2003.

[113] V. Pureza and R. Morabito. Some experiments with a simple tabu search algorithm for the manufacturer's pallet loading problem. *Computers and Operations Research*, 33(3):804–819, 2006.

[114] R. Puri and J. Gu. A BDD SAT solver for satisfiability testing: An industrial case study. *Annals of Mathematics and Artificial Intelligence*, 17(2):315–337, 1996.

[115] S. A. Rahman, P. Advani, R. Schunk, R. Schrader, and D. Schomburg. Metabolic pathway analysis web service (pathway hunter tool at cubic). *Bioinformatics*, 21(7):1189–1193, 2005.

[116] O. Ray, K. E. Whelan, and R. D. King. Logic-based steady-state analysis and revision of metabolic networks with inhibition. In *Proceedings of the 2nd International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2009)*, pages 661–666, 2009.

[117] V. N. Reddy, M. L. Mavrovouniotis, and M. N. Liebman. Petri net representations in metabolic pathways. In *Proceedings of the 1st International Conference on Intelligent Systems for Molecular Biology*, pages 328–336, 1993.

[118] A. Sabharwal, C. Ansotegui, C. Gomes, J. Hart, and B. Selman. Qbf modeling: Exploiting player symmetry for simplicity and efficiency. In *Proceedings of Theory and Applications of Satisfiability Testing (SAT 2006)*, pages 382–395, 2006.

[119] T. Schaub and S. Thiele. Metabolic network expansion with answer set programming. In *Proceedings of the 25th International Conference on Logic Programming (ICLP 2009)*, pages 312–326, 2009.

[120] S. Schuster, D. A. Fell, and T. Dandekar. A general definition of metabolic pathways useful for systematic organization and analysis of complex metabolic networks. *Nature Biotechnology*, 18:326–332, 2000.

[121] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceeding of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, 1992.

[122] B. Selman, D. Mitchell, and H. Levesque. Generating hard satisfiability problems. *Artificial Intelligence*, 81(1–2):17–29, 1996.

[123] C. E. Shannon. A symbolic analysis of relay and switching circuits. In N. Sloane and A. Wyner, editors, *Collected Papers of Claude Elwood Shannon*, pages 471–495. IEEE CS Press, 1992.

[124] C. Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *Proceedings of the Principles and Practice of Constraint Programming (CP 2005)*, pages 827–831, 2005.

[125] T. Soh, K. Inoue, M. Banbara, and N. Tamura. Experimental results for solving job-shop scheduling problems with multiple sat solvers. In *Proceedings of the 1st International Workshop on Distributed and Speculative Constraint Processing (DSCP '05)*, pages 25–38, 2005.

[126] T. Soh, K. Inoue, N. Tamura, M. Banbara, and H. Nabeshima. A sat-based method for solving the two-dimensional strip packing problem. *Fundamenta Informaticae*, 102(3,4):467–487, 2010.

[127] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear CSP into SAT. *Constraints*, 14(2):254–272, 2009.

[128] T. Tamura, K. Takemoto, and T. Akutsu. Measuring structural robustness of metabolic networks under a boolean model using integer programming and feedback vertex sets. In *Proceedings of the 2nd International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2009)*, pages 819–824, 2009.

[129] T. Tanjo, N. Tamura, and M. Banbara. Towards a compact and efficient SAT-encoding of finite linear CSP. In *Proceedings of the 9th International Workshop on Constraint Modelling and Reformulation (ModRef 2010)*. 15 pages, 2010.

[130] T. Tanjo, N. Tamura, and M. Banbara. Proposal of a compact and efficient SAT encoding using a numeral system of any base. In *Proceedings of the 1st International Workshop on the Cross-Fertilization Between CSP and SAT*. 15 pages, 2011.

[131] M. Terzer, N. D. Maynard, M. W. Covert, and J. Stelling. Genome-scale metabolit networks. *Systems Biology and Medicine*, 1(3):285 – 297, 2009.

[132] A. Tiwari, C. L. Talcott, M. Knapp, P. Lincoln, and K. Laderoute. Analyzing pathways using sat-based approaches. In *Proceedings of the Second International Conference on Algebraic Biology (AB '07)*, volume 4545 of *LNCS*, pages 155–169, 2007.

[133] C. J. Tomlin and J. D. Axelrod. Biology by numbers: mathematical modelling in developmental biology. *Nature Reviews Genetics*, 8(5):331 – 340, 2007.

[134] F. v. Harmelen, V. Lifschitz, and B. Porter, editors. *Handbook of Knowledge Representation*. Elsevier, 2008.

[135] G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10(3):253–281, 2005.

[136] T. Walsh. SAT v CSP. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP'02)*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.

[137] J. P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.

[138] L. Wei, D. Zhang, and Q. Chen. A least wasted first heuristic algorithm for the rectangular packing problem. *Computers and Operations Research*, 36(5):1608–1614, 2009.

[139] W. Wei, J. Erenrich, and B. Selman. Towards efficient sampling: exploiting random walk strategies. In *Proceedings of the 19th national conference on Artifical intelligence (AAAI 2004)*, pages 670–676, 2004.

[140] J. Whittemore, J. Kim, and K. Sakallah. SATIRE: a new incremental satisfiability engine. In *Proceedings of the 38th Conference on Design Automation (DAC '01)*, pages 542–545. ACM, 2001.

[141] L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*, ICCAD '01, pages 279–285. IEEE Press, 2001.