Maximum Satisfiability Approach to Game Theory and Network Security

廖, 暁鵬

https://doi.org/10.15017/1441264

出版情報：九州大学, 2013, 博士（工学）, 課程博士
バージョン：
権利関係：全文ファイル公表済

Kyushu University

Doctoral Thesis

Maximum Satisfiability Approach to Game Theory and Network Security

Author:
Xiaojuan Liao

Supervisor:
Prof. Ryuzo Hasegawa

A thesis submitted for the degree of
Doctor of Engineering

in the

Graduate School of Information Science and
Electrical Engineering

January 2014

Abstract

The problem of determining whether a propositional Boolean formula can be true is called Boolean Satisfiability Problem (SAT). Maximum Satisfiability Problem (MaxSAT), as well as its extensions: partial MaxSAT, weighted MaxSAT, and weighted partial MaxSAT, are optimization versions of the famous SAT problem. To date, there have been a variety of MaxSAT applications such as planning and scheduling. This thesis is concerned with a well-suited way of representing and solving real-world problems with MaxSAT, in terms of multi-agent systems and cryptographic areas.

Generally, a propositional Boolean formula is expressed in Conjunction Normal Form (CNF), which is a conjunction of clauses that are disjunctions of literals. A literal is either a positive or negative Boolean variable. Weighted partial MaxSAT (WPM) distinguishes clauses between hard and soft, where each soft clause is associated with a *positive* weight. The WPM problem is to satisfy all hard clauses and maximize the sum of weights of all the satisfied soft clauses. The fact that WPM only identifies positive weights sometimes becomes impediment to solve problems where positive and negative weights co-exist. To avoid this difficulty, in this thesis, an extended WPM (EWPM) for handling non-zero weights is presented and the relationship between EWPM and WPM solution is examined. The design of EWPM paves the way for a wider range of WPM applications.

One application of EWPM is the coalition structure generation problem (CSG), which tries to partition a set of agents into coalitions so that the total sum of payoffs of all coalitions is maximized. One of the difficulties for solving the CSG problem is that the size for representing the payoffs of possible coalitions is exponential to the number of agents. Recently, a new breakthrough has been achieved by representing the CSG problem concisely in a set of rules, where each rule is assigned to a non-zero payoff. With the compact representation schemes, the CSG problem has been significantly scaled up. This thesis provides two WPM encodings for solving the CSG problem in compact representation schemes, making use of rule relations and agent relations respectively. The rule relation-based WPM encoding is derived from the existing optimization frameworks, while the agent relation-based WPM encoding is a brand-new encoding drawing on the developed EWPM. Both encodings validate the effectiveness of the WPM solvers in solving the CSG problem, and the agent relation-based WPM encoding exhibits more desirable performance than the other one.

If all soft clauses in WPM have weight 1, the problem is regarded as partial MaxSAT. The goal of partial MaxSAT is to satisfy all hard clauses and the maximal number of soft clauses. In this thesis, the potential of partial MaxSAT is exploited for reconstructing corrupted keys of advanced encryption standard (AES), which is typically extracted from dynamic random access memory. An AES key is a series of 0-1 bits closely related to each other. The relations among key bits are naturally expressed with a set of Boolean formulas, and thus the problem of rectifying the faults in the corrupted AES key schedule is able to be formulated as a Boolean satisfiability problem. Traditionally, the AES key recovery is achieved by SAT solvers, based on the assumption that all memory bits decay from the charged state to the ground state. However, this assumption does not hold in realistic case where the majority of memory bits decay to the ground state while a small fraction of bits flip in the opposite direction. This thesis follows the realistic setting by taking the reverse flipping into consideration, and encodes the problem of AES key recovery into partial MaxSAT. Experiments demonstrate that the partial MaxSAT encoding can greatly improve the efficiency of AES key recovery from corrupted key bits.

Acknowledgements

First of all, I would like to acknowledge the China Scholarship Council (CSC) for awarding me the state scholarship, which covered all my living stipend in Japan during my Ph.D course. Thanks to this scholarship, I could fully devote myself to the research.

I would like to thank my supervisor Prof. Ryuzo Hasegawa for his support of my research and Ph.D study. Without his supervision and help, this work would not be possible. His dedication to research and patience for students will always be an example for me to follow. I gratefully thank Prof. Hiroshi Fujita for his constructive suggestions and support during my study in Hasegawa-Fujita Lab. I am heartily indebted to Prof. Miyuki Koshimura for his patient and expert guidance throughout the course of the research described in this thesis. Without the numerous discussions with him, the results presented in this thesis would never have existed. I would also like to thank Prof. Makoto Yokoo and Prof. Einoshin Suzuki for taking the time to read my thesis and giving me precious comments.

I would like to express my sincere appreciation to Prof. Makoto Yokoo and his students for their kind and generous help of my research. My sincere appreciation is extended to Dr. Tadashi Araragi for being my external advisor and providing a lot of perspicacious comments about my research. Besides, I would like to thank my collaborator Dong Hao for his valuable comments and suggestions. I would like to show my deep gratitude to Hiroshi Kihara for giving me kind help on my research. I would also like to thank Dr. Fagen Li for providing me with the valuable research knowledge and advice.

I would like to express sincere thanks to all members of Hasegawa-Fujita Laboratory. You have been remarkable source of inspiration for many ideas developed in this thesis. I am really glad to have collaborated with you all. I am grateful to my colleagues and friends at Kyushu University: Rong Huang, Leyuan Liu, Yichao Xu, Chengming Li and Hao Zhu. I am very fortunate to have met up, discussed and worked with you during my Ph.D course. Thank you for all your support on research and living

Finally I would like to thank my parents, my husband, and all my friends for their love and encouragement. Without their help and understanding, I certainly would not have been able to finish this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many problems that arise in the real world are difficult to solve partially because they present computational challenges. Furthermore, it is often important to find not just any solution to the problem, but the best one from all feasible solutions according to some objective. In this case, the problem falls into the class of optimization problems. If an optimization problem is represented as discrete variables, it is known as a combinatorial optimization problem. Maximum Satisfiability (MaxSAT), as an advanced tool for solving combinatorial optimization problem, has been applied to a wide range of practical areas.

This thesis is about the application of MaxSAT to multi-agent systems and cryptographic areas. More in particular, weighted partial MaxSAT is employed for solving the coalition structure generation problem, one of the main challenges in coalition formation. In addition, the potential of partial MaxSAT is exploited for reconstructing corrupted AES key schedule images, a series of 0-1 bits extracted from dynamic random access memory.

This chapter provides an overview of the problems that will be addressed in the rest of the thesis and gives a brief summary of the thesis contributions.

1.1 Research Area

1.1.1 SAT and MaxSAT

A *Boolean formula* is used to represent a *Boolean function*, where the definition domain and the value field are both in $B = \{Ture, False\}$. A Boolean variable takes only the

two values in B. If a Boolean variable is bounded with a Boolean value, it is said to be *assigned* a value, otherwise, it is *free*, which means it is not assigned a value. Given a Boolean formula, the problem of determining whether there exists a variable assignment that makes a Boolean formula evaluate to *true* is called the *satisfiability problem*.

A *propositional Boolean formula* is a Boolean formula that only contains logic operations *and*, *or* and *not*. Typically, a Boolean propositional formula is expressed in *Conjunctive Normal Form (CNF)*, also known as *Product of Sum (POS)* form. A formula in CNF consists of a conjunction (logic *and*) of one or more clauses. A *clause* is a disjunction (logic *or*) of one or more literals, and a *literal* is an occurrence of a Boolean variable or its negation.

A variable assignment *satisfies* a literal x if x takes the value of 1 and satisfies a literal $\neg x$ if x takes the value 0. A variable assignment satisfies a clause if it satisfies at least one literal of the clause, and satisfies a CNF formula if it satisfies all clauses of the formula.

Boolean Satisfiability Problem (SAT) determines whether there exists a variable assignment that makes a propositional Boolean formula evaluate to true. In other words, the SAT problem for a CNF formula tries to find a variable assignment that satisfies all the clauses in a Boolean propositional formula. If such an assignment exists, the formula is *satisfiable*, otherwise, the formula is *unsatisfiable*. SAT problem is the first known NP-complete problem proven by Stephen Cook in 1971 [19], and is one of the most important and extensively studied problem in computer science.

Maximum Satisfiability (MaxSAT) is a generalization of the Boolean Satisfiability problem. MaxSAT tries to find a variable assignment that satisfies the maximum number of clauses in a CNF formula. There are several variants of the MaxSAT problem: partial MaxSAT, weighted MaxSAT, and weighted partial MaxSAT. In partial MaxSAT problem for a CNF formula, some clauses are declared to be *relaxable* or *soft* and the rest are *hard*. The problem is to find a variable assignment that satisfies all the hard clauses and the maximum number of soft clauses. Weighted MaxSAT, where each clause has a bounded *positive* numerical weight, is to find a variable assignment that maximizes the sum of weights of satisfied clauses. Weighted Partial MaxSAT (WPM) is the combination of partial MaxSAT and weighted MaxSAT. In WPM, each soft clause is associated with a positive numerical value. Solving WPM corresponds to finding an assignment that satisfies all the hard clauses and maximizes the sum of weights of satisfied soft clauses (or equivalently, that minimizes the sum of weights of unsatisfied soft clauses).

1.1.2 The CSG Problem

Coalition formation is an important capacity in multi-agent systems. It is a fundamental type of interaction that involves the creation of coherent groupings of distinct, autonomous agents in order to efficiently achieve their individual or collective goals. In general, coalition formation is composed of three main activities [76]:

- Coalition structure generation: finding a coalition structure, i.e., an exhaustive set of mutually disjoint coalition, so that the performance of the entire system is optimized.
- Teamwork: optimizing the performance of each individual coalition.
- Payoff distribution: dividing the gains from cooperation among agents so as to meet certain positive or normative criteria.

Coalition structure generation (CSG) [97] is the main research issue in coalition formation of multi-agent systems. It means partitioning a set of agents into exhaustive and disjoint coalitions, where each coalition is assigned a real-valued payoff. This partition is called a *coalition structure*. Solving the CSG problem is to find a coalition structure such that the total value of all the coalitions is maximized. Two concrete examples of the CSG problem are shown as follows.

Example 1.1. *Let us consider coalition formation among three autonomous agent-robots a_1, a_2, and a_3. The payoffs of possible coalitions are listed as follows.*

- *If a_1 works alone, the payoff of 1 is achieved.*
- *If a_2 works alone, the payoff of 0 is achieved.*
- *If a_3 works alone, the payoff of 0 is achieved.*
- *If a_1 and a_2 work together, the payoff of 1 is achieved.*
- *If a_1 and a_3 work together, the payoff of 1 is achieved.*
- *If a_2 and a_3 work together, the payoff of 1 is achieved.*
- *If a_1, a_2 and a_3 work together, the payoff of 2 is achieved.*

It indicates that a_1 is able to secure the payoff of 1 if acting alone, while a_2 and a_3 produce nothing if performing separately. Besides, no pair of cooperating agents achieves any value added except a_2 and a_3 who are able to lead to the payoff of 1 if they work together. There are two optimal partitions of these three agents. The first one is the

grand coalition, i.e, a_1, a_2 and a_3 perform cooperatively. The other one is to assign a_2 and a_2 to the same group while let a_1 work alone. The total values of these two partitions are both 2.

Example 1.2. *Let us consider coalition formation among the three autonomous agent-robots a_1, a_2, and a_3 again. The payoffs of coalitions are almost the same as that in Example 1.1, except that the following condition is incorporated.*

- *The performance of a_1 improves by an additional 0.5 when facing the cooperation of a_2 and a_3.*

In this case, the value of coalition $\{a_1\}$ is different in $\{\{a_1\}, \{a_2, a_3\}\}$ than in $\{\{a_1\}, \{a_2\}, \{a_3\}\}$. When considering the payoff of $\{a_1\}$, the formation of other coalitions should be taken into account, i.e, whether a_2 and a_3 are assigned to the same coalition.

These two examples provide an overview of two types of the CSG problem. In Example 1.1, the value of a coalition is independent of how other coalitions are formed, while in Example 1.2, the value of a coalition is affected by the formation of other coalitions. This effect is known as *externality from coalition formation*. Clearly, the CSG problem with externalities is more complicated that the problem without externalities as more possible coalitions are needed to be examined. Currently, the majority of works have been devoted to solving the CSG problem without externalities, while only a few works have been designed for solving the problem with externalities.

The difficulty of solving the CSG problem lies in the exponentially increasing number of possible solutions (i.e., coalition structures), with $\mathrm{O}(n^n)$ for n agents. Furthermore, the representation size for enumerating the payoffs of possible coalitions is also considerably large, $\mathrm{O}(2^n)$ for CSG without externalities, and $\mathrm{O}(n^n)$ for CSG with externalities [87]. To date, the state-of-the-art algorithms are able to solve the CSG problem with only a dozen of agents.

The large representation size of the CSG problem could be alleviated by employing compact representation schemes. To date, a variety of such schemes have been developed, e.g., marginal contribution nets (MC-nets) [43], synergy coalition groups (SCGs) [18], SCGs in multi-issue domains [17], and embedded MC-nets [69]. In these concise representation schemes, the CSG problem is expressed in a set of rules, in the form of *condition* $\rightarrow$ *value*, which could represent all possible mappings between coalitions and the corresponding values. Thus solving the CSG problem amounts to maximizing the

sum of values of rules where the corresponding conditions satisfy some constraints. With these concise representation schemes, the CSG problem can scale up significantly under some off-the-shelf optimization algorithms. If the weighted rules and constraints are respectively encoded into soft clauses and hard clauses, the CSG problem is formulated as a weighted partial MaxSAT problem. The only obstacle to adopt MaxSAT solvers is that the value of a coalition is either positive or negative, while the weight in weighted partial MaxSAT must be positive.

1.1.3 Recovering AES Keys from a Cold Boot Attack

Dynamic random access memory (DRAM) is the main memory used in most modern computers. Most security practitioners assumed that a computer's memory is erased almost immediately when it loses power, or that whatever data remains is difficult to retrieve without specialized equipment. This assumption has been declared incorrect in 2008 [34]. The observed fact is, after power is lost, DRAM retains its contents for several seconds at room temperature, and for minutes or even hours if the chips are kept at low temperature. This feature of DRAM is called *DRAM remanence*. Residual data can be recovered using simple, nondestructive techniques that require only momentary physical access to the machine.

Encryption systems typically store the encryption keys in RAM to speed up the retrieval of keys, which opens a door to attackers to retrieve the encryption keys by cutting power to the memory. If attackers are forced to cut power to the memory for too long, the contents of memory will become corrupted. Confronting with the corrupted keys, attackers have developed algorithms to correct errors in private and symmetric keys.

Advanced Encryption Standard (AES) is a worldwide prevalent symmetric cryptographic algorithm nowadays. The recovery of AES keys is usually achieved by exploiting the redundancy of key material inherent in cryptographic algorithms. An AES key refers to a key schedule consisting of multiple round keys. According to the AES key expansion algorithm [31], all these round keys are computed from an initial key that is relatively short compared to the entire key schedule. This implies that the round keys contain a large amount of redundancy. In other words, bits in the round keys are so strongly constrained with each other that an attacker may recover the entire key schedule even if he knows only some parts of the key bits. Besides, the relations that have to be satisfied between the round key bits are naturally expressed in a set of Boolean formulas, which could be reformulated as Boolean propositional formulas. Therefore, it is possible to

employ SAT or MaxSAT solvers to recover the AES key schedule, as long as a moderate amount of key bits is available. The occasions of employing SAT or MaxSAT solvers are distinguished by the different assumptions adopted in the key recovery.

According to the observation of DRAM remanence, the probability of decaying to the ground state approaches 1 as time goes on, while the probability of flipping in the opposite direction remains relatively constant and small (around 0.1% observed in [34]). In other words, given a section of memory contents after power is removed, most bits in the charged state that remain in the memory are correct because the probability of flipping from the ground state to the charged state is quite tiny. This tiny probability is sometimes neglected and thus all the memory bits in the ground state after extracted from memory are correct without any exception. This is called *perfect assumption*, on which based the key recovery problem is modeled as a SAT problem. On the other hand, if the reverse flipping is taken into consideration, recognized as *realistic assumption*, the problem can be modeled as partial MaxSAT, in which the probabilistically correct bits are modeled as soft clauses.

1.2 Thesis Contribution

This thesis contributes to applying MaxSAT to solve the CSG problem and recover AES key schedules by providing added values to the following points:

Extending WPM to deal with negative weights. In weighted paritial MaxSAT (WPM), it is natural to limit the weights of soft formulas in the positive domain because the original intention of MaxSAT is maximizing the number of satisfied formulas. However, there are some applications expressed as formulas with both positive and negative weights. For example, in the CSG problem that are represented by a set of rules, the values of these rules can be both positive and negative. In this context, we have extended the WPM to deal with negative weights, referred to as extended WPM (EWPM), thus any propositional Boolean formulas with non-zero weights could be handled by an off-the-shelf MaxSAT solver. Moreover, in order to solve EWPM instances with WPM solvers, we have presented transformation from EWPM to WPM and examine the relationship between EWPM and WPM solutions.

Solving the CSG problem with WPM. Traditionally, to represent a characteristic function or partition function in the CSG problem, a naive solution is to enumerate the payoffs to each set of agents, thus requiring space exponential to the number of agents

in the game. This is prohibitive when the number of agents is large. In order to conquer the problem of exponentially growing representation size, recent works have made use of compact representation schemes and scaled up the CSG problem significantly.

Inspired by the recent works that employ the concise representation schemes, we have encoded the CSG problem into Boolean propositional formulas, which could be handled by existing WPM solvers. Specifically, our encodings are composed of two methods. The first one, named rule relation-based WPM encoding, is directly derived from the existing works [75, 111]. The results obtained confirm the effectiveness of our encoding and show that WPM could be successfully applied to the CSG problem. Our second method, called agent relation-based WPM encoding, managed to further improve the performance, which saves the computation time by more than half compared to the rule relation-based WPM encoding.

Recovering AES key schedules with MaxSAT. In a true cold boot attack observed by Halderman [34], with time goes on after power is removed, most memory bits decay from the charged state to the ground state, with only a small fraction, around 0.1%, flip to the charged state. This is the *realistic assumption* for a cold boot attack. However, in many previous works, the decay direction is assumed only from the charged state to the ground state with no bit flipping in the opposite direction. We call it *perfect assumption*. In this thesis, we have extended the previous SAT-based approach [49], which could only recover AES keys under perfect assumption, to adapt the realistic assumption. Moreover, we have recast the problem of AES key recovery under the realistic assumption as a partial MaxSAT problem. Specifically, all bits in the charged state are encoded as soft clauses, and the relations that have to be satisfied between different round key bits are encoded as hard clauses. Experiments show that the MaxSAT approach is significantly superior to the SAT approach, which is tuned from the previous work [49].

1.3 Thesis Organization

The organization of this thesis is as follows. Chapter 2 provides an introduction to the preliminaries used in the remainder of the this thesis, including concepts related to MaxSAT, various techniques used for MaxSAT solving, and encodings that transform from a propositional formula to CNF formula. MaxSAT solving techniques play a crucial role to improving the efficiency of problem solving, and the choice of CNF encoding is as

important as that of MaxSAT solving algorithms, since currently, many MaxSAT solvers are designed to solve problems represented typically in CNF formulas.

In Chapter 3, the standard weighted partial MaxSAT (WPM) is extended for handling not only positive weights but also negative weights. The original intension of the extension is to describe the real-world problems that are associated with both positive and negative values, and then employ the off-the-shelf WPM solvers to these problems. To this end, this chapter first shows the way of transforming from extended WPM (EWPM) to the standard one, and provides a rigorous proof on the relation between EWPM and WPM solutions.

Chapter 4 presents a WPM encoding on solving the coalition structure generation (CSG). The encoding provided in this chapter is directly derived from the previous work by Yokoo et. al [111]. First an overview of the previous work that is the most related to our encodings is provided, which has been shown sound and more efficient than other works. This forms the basis for the WPM encoding discussed subsequently. A procedure to encode the previous work into WPM formulas is provided, including the encoding of the basic CSG problem as well as its extension. Experiments are used to show the efficiency and scalability of the WPM encoding.

Chapter 5 provides a brand-new WPM encoding for the CSG problem, taking advantage of the EWPM-to-WPM transformation described in Chapter 3. The notion of agent relations is introduced and the encodings of the CSG problem based on agent relations are defined. In the rest of this chapter, the WPM encoding towards solving the CSG problem with positive values and negative values are discussed step by step. Experimental data and comparison results are provided to validate the effectiveness of the proposed encoding.

In Chapter 6, two propositional logical encodings for recovering AES key schedules is provided. There are two different assumptions for key recovery, i.e., perfect assumption and realistic assumption. Perfect assumption assumes all memory bits tend to decay to the ground state after power is removed, while in the realistic assumption, the phenomenon of decaying to the ground state and flipping to the charged state may co-exist. The works for recovering the AES keys under different assumptions are analyzed. Since the realistic assumption is more suitable for the real-world case, this chapter presents two approaches for recovering AES keys under realistic assumption, with SAT and partial MaxSAT solvers respectively. Experiments are provided to demonstrate the effectiveness of the proposed approaches.

Finally, Chapter 7 contains a summary of this thesis and a discussion of some future research directions that may be worth exploring.

Chapter 2

Preliminary

The purpose of this chapter is to set the basic concepts and notations of the Maximum Satisfiability (MaxSAT) problem and game theory. This chapter consists of four sections. The first section introduces the basic formal concepts on SAT, MaxSAT, partial MaxSAT, weighted partial MaxSAT, and weighted partial MaxSAT. In Section 2.2, the algorithms for solving MaxSAT are outlined. The third section describes the techniques for transforming from a Boolean propositional formula to Conjunction Normal Form (CNF). Finally, in Section 2.4, an overview of game theory is provided.

2.1 SAT, MaxSAT and Its Extension

A *Boolean formula* is a string that represents a *Boolean function*. A Boolean function is a function of the form: $B^k \rightarrow B$, where $B = \{True, False\}$ is a Boolean domain and k is the arity of the function. Usually $True$ and $False$ are represented by 1 and 0 respectively. A *propositional Boolean formula* is a Boolean formula that only contains logic operations *and*, *or* and *not* (sometimes called *negation* or *complement*), symbolized as $\wedge$, $\vee$ and $\neg$, respectively. Some examples of propositional Boolean formulas are listed as follows:

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

A Boolean propositional formula can be expressed in *Conjunctive Normal Form (CNF)*, also known as *Product of Sum (POS)* form. A formula in CNF consists of a conjunction (logic *and*) of one or more clauses. A *clause* is a disjunction (logic *or*) of one or more literals, and a *literal* is an occurrence of a Boolean variable or its negation. An example of a propositional Boolean formula in CNF is $\phi = (b \vee \neg c) \wedge (a \vee \neg c) \wedge b$, which is also can be represented by a set of clauses as $\phi = \{b \vee \neg c, a \vee \neg c, b\}$, or an assemble of literal sets like $\phi = \{\{b, \neg c\}, \{a, \neg c\}, \{b\}\}$. In this thesis, a CNF formula is denoted by a set of clauses, i.e., conjunction is omitted.

The problem of determining whether there exists a variable assignment that makes a propositional Boolean formula evaluate to true is called *Boolean Satisfiability Problem (SAT)*. In other words, the SAT problem tries to find a variable assignment to a CNF formula that satisfies all the clauses in a Boolean propositional formula. If such an assignment exists, the formula is *satisfiable*, otherwise, the formula is *unsatisfiable*.

Example 2.1. *$\phi = \{a \vee b \vee \neg c \vee d, a \vee \neg b \vee \neg c, a \vee b, \neg d, d\}$ is a Boolean propositional formula in CNF. This formula contains five clauses: $(a \vee b \vee \neg c \vee d)$, $(a \vee \neg b \vee \neg c)$, $(a \vee b)$, $(\neg d)$, and (d). The first clause $(a \vee b \vee \neg c \vee d)$ contains four literals, i.e., a, b, $\neg c$, and d. This formula is unsatisfiable because the last two clauses are conflicting, which means they cannot be satisfied at the same time.*

SAT problem is the first known NP-complete problem proven by Stephen Cook in 1971 [19] and independently by Leonid Levin in 1973. The proof shows how every decision problem in the complexity class NP can be reduced to the SAT problem for CNF formulas. That the SAT problem is NP problem briefly means that there is no known algorithm that efficiently solves all instances of SAT, and it is generally believed that no such algorithm can exist. A class of algorithms to efficiently solve a large enough subset of SAT instances is called *SAT solver*, which is useful in various practical areas such as circuit design and automatic theorem proving, as well as solving problems in computer science. Extending the capabilities of SAT solving algorithms is an ongoing area of progress. However, no current such methods can efficiently solve all SAT instances.

The last decade has witnessed a dramatic speed-up of SAT solvers: problems with thousands of variables are now solved in milliseconds by state-of-the-art SAT solvers. One of the main reasons for such remarkable improvements is the wide range of SAT applications. Examples include software verification [16, 44], model checking of finite-state systems [11, 102], AI planning in artificial intelligence [91, 99], as well as cryptographic

areas that have been on the rise in recent years [49, 80]. In addition to practical applications, the extension of SAT has also attracted much attention, such as Satisfiability Modulo Theories (SMT) [7], Quantified-Boolean Formulas (QBF) [56], and Maximum Satisfiability (MaxSAT) [38, 58]. Readers may refer to [66] for more applications of SAT.

Given a Boolean propositional formula, if it is unsatisfiable, SAT solvers only report that no solution exists, without any information on unsatisfiable instances. However, assignments violating a minimum number of constraints, or satisfying all the compulsory (*hard*) constraints and as many optional (*soft*) constraints as possible, can be considered as acceptable solutions in real-life scenarios. To cope with this limitation of SAT, MaxSAT and its extensions, such as partial MaxSAT and weighted MaxSAT, are becoming an alternative for representing and efficiently solving over-constrained problems [12].

The MaxSAT problem for a CNF formula is the problem of finding a variable assignment that maximizes the number of satisfied clauses. MaxSAT is often used to mean Min-UNSAT, because finding an assignment that maximizes the number of satisfied clauses is equivalent to finding an assignment that minimizes the number of unsatisfied clauses. MaxSAT is useful to measure the extent of unsatisfiability of a CNF formula.

Three extensions of MaxSAT are more well-suited for representing and solving over-constrained problems: partial MaxSAT, weighted MaxSAT, and weighted partial MaxSAT.

In a partial MaxSAT instance, each clause is labeled either *hard* or *soft*. The hard clauses must be obligatorily satisfied, while the soft clauses can be unsatisfied. The goal of solving the partial MaxSAT instance is to satisfy all hard clauses and the maximal number of soft clause. The partial MaxSAT problem is easily extended to a SAT problem if all the clauses are hard, and a MaxSAT problem if all the clauses are soft.

Example 2.2. *Given a partial MaxSAT instance* $\phi = \{[a], [\neg a \vee \neg b], (b \vee c)\}$*, the first and second clause, enclosed by "[ ]", are hard clauses, and the third clause, enclosed by "( )", is a soft clause. To satisfy the hard clauses, a and b are forced to be 1 and 0 respectively. Based on the assignment of a and b, the true assignment of c has to be 1 so that the soft clause can be satisfied.*

A weighted MaxSAT instance is expressed in weighted CNF, where each clause is assigned a positive integer. The problem is to find a truth assignment that maximizes the sum of weights of satisfied clauses. In a special case, if the weights of all the clauses in weighted MaxSAT is equal to one, the problem is regarded as a MaxSAT problem.

Example 2.3. *A weighted MaxSAT instance* $\phi = \{(a, 2), (\neg a \vee \neg b, 3), (b \vee c, 4)\}$ *has three weighed clauses holding weights 2, 3, 4 respectively. All of the three clauses can be satisfied by assigning 1, 0, 1 to a, b, c respectively, which leads to the maximal sum of weights of satisfied clauses.*

The weighted partial MaxSAT (WPM) is the combination of partial MaxSAT and weighted partial MaxSAT. A WPM instance distinguishes hard and soft clauses, where each soft clause is assigned a positive integer. Solving the WPM instance is to satisfy all hard clauses and maximize the sum of weights of satisfied soft clauses. The definition of WPM can be easily extended to partial MaxSAT, where all soft clauses have weight 1, and weighted MaxSAT, where no clauses are hard.

Example 2.4. *Given a WPM instance* $\phi = \{[a], [\neg a \vee \neg b], (b \vee c, 2), (b \vee \neg c, 7)\}$*, the first and second clause, enclosed by "[ ]", are hard clauses, while the third and fourth clauses, enclosed by "( )", are soft clauses. To satisfy the hard clauses, a and b are forced to be 1 and 0 respectively. Based on the assignment of a and b, the true assignment of c is preferred to be 0 so that the weight 7 can be earned, which is larger than the other case that the gain is merely 2.*

Many important problems can be naturally expressed as MaxSAT, including academic problems such as Max-Cut or Max-Clique, as well as problems from many industrial domains. Concrete examples include the following domains: routing problems [115], hardware debugging [15, 63, 95], software debugging [46, 47], scheduling [51, 113], planning [20, 48, 92, 116], probabilistic reasoning [78], electronic markets [96], etc. Additionally, many problems originally formulated in other optimization frameworks can be easily reformulated as MaxSAT, such as the Pseudo-Boolean Optimization framework [1], the Weighted CSP framework [55] and the MaxSMT framework [74]. Readers may refer [71] for more applications of MaxSAT.

2.2 MaxSAT Algorithms

The last two decades have witnessed significant progress in the development of theoretical, logical and algorithmic aspects of MaxSAT solving, as well as new and appealing research directions such as exploring the impact of modeling on the performances of MaxSAT solvers [6]. Early theoretical MaxSAT research provided insights in the complexity of the problem [9, 77]. Moreover, the MaxSAT evaluation [28], which was firstly

held in 2006, plays as a driving force for motivating the development of novel MaxSAT technologies.

Generally speaking, there are two approaches for MaxSAT solving techniques: approximation algorithms that compute near-optimal solutions, and exact (or complete) algorithms that compute optimal solutions.

Heuristic local search algorithms are the foundation of early practical works to find near-optimal solutions. Whereas many exact MaxSAT solvers use a local search algorithm to compute an initial assignment of variables, these algorithms do not guarantee to find the optimal solution. That is why this class is referred to as incomplete solvers. The approximation of a MaxSAT solution is usually measured by a factor that is bounded by a constant α or a slowly growing function of the input size. Given a constant α, an algorithm is α-approximation for a maximization problem if it provides a feasible solution in polynomial time which is at least α times the optimum, considering all the possible instances of the problem [12]. A number of improvements have been achieved on the performance guarantee, from 1/2, proposed in 1974 [45], to 0.7584, proposed in 1995 [33]. Later on, a limit on approximability was proved by Håstad [42] that unless NP=P, no approximation algorithm for MaxSAT can achieve a performance guarantee better than 7/8. This theory was proved again in [50], showing that the constant 7/8 is tight. Recently, semidefinite programming has been shown quite promising for approximating MaxSAT solutiosn. Readers may refer to [3] to learn more about how to approximate MaxSAT with semidefinite programming.

The exact algorithms can be classified into two approaches. The one follows a branch and bound (BB) algorithm and applies several techniques tailored to MaxSAT. Another one makes use of a state-of-the-art SAT solver as an inference engine, referred to as SAT-based approach.

Many contemporary exact MaxSAT solvers follow a BB algorithm [2, 13, 24, 39, 54, 62, 82, 83, 114], which ensures the minimal number of unsatisfied clauses in a MaxSAT problem. Given a MaxSAT instance ϕ, BB explores a search tree that represents the space of all possible assignments for ϕ in a depth-first manner. At every node, BB compares the upper bound (UB) with the lower bound (LB). UB is the best solution (i.e., the minimum number of falsified clauses) found so far for a complete assignment, and LB is the sum of the number of clauses which are falsified by the current partial assignment plus an underestimation of the number of clauses that will become unsatisfied f the current partial assignment is completed. If $LB \geq UB$, the algorithm prunes the

subtree below the current node and backtracks chronologically to a higher level in the search tree. If $UB < LB$, the algorithm tries to find a better solution by extending the current partial assignment by assigning one more variable. The value of UB after the search of entire tree is the optimal number of unsatisfied clauses in ϕ.

SAT-based approach uses a SAT solver to iteratively search for satisfiable subsets of clauses within certain constraints. SAT-based solvers are further classified into two categories: satisfiability-based [10] and unsatisfiability-based [5, 64].

For a satisfiability-based solver, given a MaxSAT instance $\phi = \{C_1, \ldots, C_n\}$, a new variable b_i is added to each clause C_i $(1 \leq i \leq n)$. b_i is called a *blocking variable*. Solving the MaxSAT problem for ϕ is to minimize the number of blocking variables that evaluate to true, called *true blocking variables*, in $\phi' = \{C_1 \vee b_i, \ldots, C_n \vee b_n\}$. The minimal satisfied assignment is searched by iterative calls to a SAT solver, summarized as follows [52]. First run the SAT solver on ϕ' without any constraints to get an initial model and count the number k of true blocking variables in the model, then add a constraint to limit the number of true blocking variables to less than k, and run the solver again. If the problem is unsatisfied, k is the optimal solution. Otherwise, the process is repeated with the constraint that limits the number of true blocking variables to a smaller integer. This process terminates when the problem becomes unsatisfied.

For an unsatisfiability-based solver, given a MaxSAT instance ϕ, the following process is iterated until ϕ is satisfiable: First run a SAT solver on ϕ. If ϕ is unsatisfiable, extract an unsatisfiable subset $US = \{C_1, \ldots, C_m\}$ from ϕ and introduce m new blocking variables b_i $(1 \leq i \leq m)$. Then, replace C_i with $C_i \vee b_i$ $(1 \leq i \leq m)$ and add a constraint $\sum_{i=1}^{m} b_i = 1$ to build a new ϕ. If ϕ is satisfiable, the iteration terminates. The number of iterations indicates the number of falsified clauses in the original ϕ.

2.3 CNF Encodings

Before a combinational problem or combinational optimization problem can be solved by SAT or MaxSAT solvers, it must usually be transformed to CNF, which has the advantage of being a very simple form, leading to easy algorithm implementation and a common file format. However, after being transformed to CNF, the formula may lose a great deal of structural information. To conquer this drawback, solvers for non-CNF problems have been devised [72, 81, 105, 106]. These techniques can yield great improvements on certain structured problems but CNF currently remains the norm [12].

This section describes techniques for transforming from a propositional formula to CNF, including transformation by Boolean algebra and Tseitin encoding. A propositional formula is a formal expression that denotes a proposition, which is a statement telling either true or false.

2.3.1 Transformation by Boolean Algebra

Boolean algebra is the subarea of algebra in which the values of the variables are the truth values *true* and *false*. It is fundamental in the development of computer science and digital logic, as well as in set theory and statistics [35].

The main operations in Boolean algebra are *and*, *or*, and *not*. These basic operations can be taken as a basis for other derived Boolean operations that can be built up from them by *composition*, the manner in which operations are combined or compounded. Some examples of derived operations composed from the basic operations are shown as follows.

$$x \to y = \neg x \vee y$$
$$x \oplus y = (x \vee y) \wedge \neg (x \wedge y)$$
$$x \equiv y = \neg (x \oplus y)$$

The operation $x \to y$ is called *material implication*. If x is true then the value of $x \to y$ is taken to be that of y. However, if x is false, the value of $x \to y$ is constantly true regardless of that of y. $x \oplus y$ is called *exclusive or*. It excludes the possibility of x and y taking the same truth value. In other words, the value of $x \oplus y$ is true just when x and y have different truth values. $x \equiv y$, the complement of exclusive or, is *equivalence* or *Boolean equality*. The value of $x \equiv y$ is true just when x and y have the same value.

A propositional formula can be transformed to a logically equivalent CNF formula by using the rules of Boolean algebra. Take the example of a propositional formula [107], shown as follows.

Example 2.5. *Let us consider the following propositional formula:*

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

The implications can be decomposed:

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

The conjunctions and disjunctions can be rearranged:

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

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

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

Finally, subsumed clauses can be removed, leaving the conjunction shown as follow:

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

This propositional formula finally reduces to a compact CNF formula, but in general this method generates exponentially large formulas [12].

2.3.2 Transformation by Tseitin Encoding

The naive approach to use rules of Boolean algebra results in an exponential increase in equation size. To avoid the exponentially large size, conversion to CNF is usually achieved by using the well-known Tseitin encoding [108], which outputs an equation whose size grows linearly relative to the input.

Tseitin encoding generates a new variable, (or say, auxiliary variable), for each subformula of the original formula, as well as a set of clauses that capture the equivalent relationship [1] between these new variables and the subformulas. The procedure of Tseitin encodings is exhibited in the following example.

Example 2.6. *Consider the propositional formula in example 2.5. A new Boolean variable f_1 is introduced to replace the subformula $(c \wedge d)$, with the definition:*

$$f_1 \leftrightarrow (c \wedge d)$$

[1] Formula ϕ and ϕ' are equivalent means ϕ is satisfiable if and only if ϕ' is satisfiable.

This definition is reduced to clauses:

$$(\neg f_1 \vee c) \wedge (\neg f_1 \vee d) \wedge (\neg c \vee \neg d \vee f_1)$$

Similarly another new Boolean variable f_2 *is introduced to replace the subformula* $(c \wedge e)$*, with the definition*

$$f_2 \leftrightarrow (c \wedge e)$$

which is reduced to clauses

$$(\neg f_2 \vee c) \wedge (\neg f_2 \vee e) \wedge (\neg c \vee \neg e \vee f_2)$$

Then the original formula is reduced to

$$(\neg a \vee f_1) \wedge (\neg b \vee f_2)$$

Next, two more variables are introduced with definitions

$$f_3 \leftrightarrow (\neg a \vee f_1)$$

and

$$f_4 \leftrightarrow (\neg b \vee f_2)$$

and clauses

$$(\neg f_3 \vee \neg a \vee f_1) \wedge (a \vee f_3) \wedge (\neg f_1 \vee f_3) \wedge (\neg f_4 \vee \neg b \vee f_2) \wedge (b \vee f_4) \wedge (\neg f_2 \vee f_4)$$

The formula is now reduced to

$$(f_3 \vee f_4)$$

This formula is already in clausal form as we desired.

Tseitin encoding is linear in the size of the original formula as long as the Boolean operators that appear in the formula have linear clausal encodings. The operators *and*, *or*, *not*, and *implies* all have linear clausal encodings, while encoding *exclusive or* requires exponential number of clauses to that of variables. We will introduce the CNF encoding of *exclusive or* in Chapter 6.

2.4 Game Theory

Game theory, which is primarily used in economics, describes multi-person decision scenarios where each player chooses actions that result in the optimal possible rewards for self, while anticipating the rational actions from other players. It provides rich mathematical tools for resolving multi-criteria optimization problems among multiple entities who behave strategically. Game theory has been widely recognized as an important tool in many fields, including economics, political science, and psychology, as well as logic and biology. In the area of computer science, there have been also a large number of applications, such as job scheduling, cryptology, network security, sensor networks, etc.

Formally, a *game* is the collection of the following definitions:

- *Player.* A player is the basic entity of a game who makes decision and then takes actions.
- *Action.* An action constitutes a move in the given game.
- *Strategy.* A strategy is a plan of action that a player can take during the game.
- *Payoff (Outcome).* A payoff is a positive or negative reward to a player for a given action within the game.

2.4.1 Non-cooperative game

A non-cooperative game is one in which players make rational decisions independently, in the aim of maximizing their individual payoffs. A solution concept of a non-cooperative game is called *Nash equilibrium*, which describes a steady state condition of the game. At the steady point, no player would prefer to change his strategy as that would lower his payoffs given that all other players are adhering to the prescribed strategy.

A canonical example of a non-cooperative game is *prisoner's dilemma*, which shows two individual players might not cooperate, even if it appears that being cooperative is their best choice. The scenario of prisoner's game is presented as follows. The police arrest and imprison two persons (suppose A and B) but do not have enough evidence to convict them. Then the police separate the two prisoners and provide each prisoner with the opportunity either to betray the other, by testifying that the other committed the crime, or to cooperate with the other, by remaining silent. Here's how it goes:

- If A and B both betray the other, each of them serves 2 years in prison.

- If A betrays but B remains silent, A will be set free and B will serve 3 years in prison (and vice versa).
- If A and B both remain silent, both of them will only serve 1 year in prison (on the lesser charge).

In this scenario, since the two prisoners will have no opportunity to reward or punish their partner in future, they need to only care their current choices. Obviously, betraying a partner offers a greater reward than cooperating with them, no matter which action his partner takes. Therefore, rational self-interested prisoners would betray the other, and the outcome is the both prisoners serve 2 years in prison. This is the Nash equilibrium of the game.

2.4.2 Cooperative game

A cooperative game is a game where groups of players may enforce cooperative behaviour, hence the game is a competition between coalitions of players. In a non-cooperative game, any cooperation among players must be self-enforcing, while in a cooperative game, the cooperation is guaranteed by a binding agreement. In prisoner's dilemma, if the two prisoners' behaviours are enforced by a third party, they can achieve the optimal outcome, i.e., prisoners cooperate with each other by remaining silent.

A cooperative game is given by specifying a value for every possible set of players, known as a *coalition*. Formally, the game (coalitional game) consists of a finite set of players A, called the *grand coalition*, and a characteristic function $v : 2^A \to \mathbb{R}$ from the set of all possible coalitions to a set of payments that satisfies $v(\emptyset) = 0$. The function describes how much collective payoff a set of players can gain by forming a coalition. The players are assumed to choose which coalitions to form, according to their estimate of the way the payment will be divided among coalition members. Coalitional games have been proven highly influential in the research of multi-agent systems, composed of multiple interacting intelligent agents (agents are referred to as players in game theory domain). Multi-agent systems can be used to solve problems that are difficult or impossible for an individual agent or a monolithic system to solve.

Coalition structure generation (CSG) is one of the main research issues in the use of coalitional games in multi-agent systems. It is related to the problem that how to maximize the social welfare, that is, the total value of coalitions, among agents. Solving

the CSG problem is one of the main contributions devoted in this thesis, which will be elaborated in Chapter 4 and Chapter 5.

2.4.3 Game Theory Applied in Network Security

Conventional cryptography and intrusion detection systems provide the first line of the defense in networks. However, in the presence of inside attackers and complicated attacks, the standard cryptography may be crippled and the traditional intrusion detection systems may be confused. In this context, as game theory has the ability to deal with problems where multiple players with contradictory objectives compete with each other, it can provide us with a mathematical framework for analysis and modeling network security problems. To date, many existing game-theoretic research as applied to network security falls under non-cooperative games, with interactions between the administrator (or legitimate user responsible for detection in self-organized networks) and an attacker. Sometimes the game could be a cooperative game where the players cooperatively achieve the same goal, e.g., to cooperatively detect the malicious around them.

A structured and comprehensive overview of the research contributions that analyze and solve security and privacy problems in computer networks by game-theoretic approaches is illustrated in [65]. Other works [30, 94] associate the network security with game theoretic approaches according to different game types, namely, the static or dynamic game with the complete or incomplete information. These works provide comprehensive understanding of game theoretic solutions on cyber security problems. In order to summarize which kind of attacks is suitable to be thwarted by game theoretical approaches, efforts have been devoted to investigating the association between network attacks and game theory. Our previously published works [61] present a classification which sorts a variety of attacks in wireless ad hoc networks into two types, namely *palpable attack* and *subtle attack*. Palpable attacks result in conspicuous impacts on network functions, which are intolerable to users, while subtle attacks lead to invisible damages in vaguer way. Game models, especially in game players, regularly vary according to the two types. For palpable attacks, game can be directly played between attackers and normal users, but for subtle attacks, game usually draws support from additional player or mechanisms to help detection. In addition, both palpable attacks and subtle attacks are possible to be eliminated by proactive cooperation of legitimate users.

It is worth noting that, although the proactive cooperation of legitimate users is useful to detect attacks, in resource-starved environment such as wireless ad hoc networks,

the proactive cooperative detection on abnormal behaviors is difficult to achieve, as saving energy is placed high priority. In this situation, effective incentive mechanisms to stimulate cooperation is of great importance. Take the Sybil attack [25] as an example. A common method to detect Sybil nodes is resource testing [57]. The conventional resource test includes computation, storage, communication [25] and radio resource test [73]. More recently, psychometric tests and color tests were proposed to identify Sybil groups, based on the fact that Sybil identities forged by one user share the same personal psychometric nature [36]. However, these intended resource tests have side effects on wireless ad hoc networks due to the limited resource on each node. If nodes spend too much resource on testing, the performance of normal communications would be affected. At this point, many solutions employ local detection to capture misbehavior and then enhance the detection accuracy with information exchange. Encouraging users to share reliable detection information could be achieved by a repeated game among users, which helps utility-driven users to share information with reliable neighbors [60].

Chapter 3

Extending MaxSAT to Deal with Negative Weights

The Maximum Satisfiability (MaxSAT) problem is an extension of Satisfiability (SAT) that is able to represent optimization problems. Many optimization problems can be expressed as MaxSAT, such as electronic design automation (EDA), planning, probabilistic inference, and software upgradeability.

In some real-world optimization problems, some constraints may be more important to satisfy than others, then the MaxSAT problem can be extended by assigning different positive weights to different constraints. In this case, it is natural to cast MaxSAT in terms of maximizing the total weights of the satisfied constraints.

This chapter presents an extended weighted partial MaxSAT (EWPM) to deal with negative weights, so that the existing weighted partial MaxSAT (WPM) solver could be applied to situations where the weights involved in a problem are both positive and negative. Specifically, Section 3.1 gives the definition of the standard WPM formula. Section 3.2 presents EWPM for handling negative weights. In order to solve EWPM instances with the existing WPM solver, an EWPM-to-WPM transformation is also provided, followed by the investigation of the relationship between EWPM and WPM solutions. Finally, Section 3.3 summarizes this chapter.

3.1 Weighted Partial MaxSAT

A *weighted clause* is a pair (C, w), where C is a clause and w, its weight, is a positive integer or infinity. A clause is called *hard* if its weight is infinity, otherwise it is *soft*. A *weighted partial MaxSAT* (WPM) instance is a multiset of weighted clauses $\phi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$, where the first m clauses are soft and the last m' clauses are hard. A *truth assignment* of ϕ is a mapping that assigns to each variable in ϕ either 0 or 1.

Given a WPM instance ϕ and a truth assignment I, the *benefit* of I on ϕ, noted $\mathit{benefit}(\phi, I)$, is the sum of the weights of the soft clauses satisfied by I if I satisfies all the hard clauses, otherwise, $\mathit{benefit}(\phi, I) = -\infty$. ϕ is *satisfiable* if ϕ has a truth assignment which satisfies all the hard clauses, otherwise it is *unsatisfiable*. The WPM problem for a WPM instance ϕ is the problem of finding an optimal assignment I of ϕ that maximizes $\mathit{benefit}(\phi, I)$, that is, $\mathit{benefit}(\phi, I) \geq \mathit{benefit}(\phi, I')$ for an arbitrary assignment I'.

3.2 Extended Weighted Partial MaxSAT

In WPM, it is natural to limit the weights of soft clauses in the positive domain because the original intention of MaxSAT is *maximizing the number* of satisfied clauses. However, there are problems which we express as formulas with both positive and negative weights naturally. This section presents the standard extension of WPM and examine the relationship between solutions of WPM and its extension.

3.2.1 EWPM-to-WPM Transformation

As an extension of WPM, an *Extended weighted partial MaxSAT* (EWPM) instance is a multiset of weighted formulas $\phi = \{(F_1, w_1), \ldots, (F_m, w_m), (F_{m+1}, \infty), \ldots, (F_{m+m'}, \infty)\}$, where the first m formulas are soft and the last m' formulas are hard. A *weighted formula* is a pair (F, w), where F is a propositional Boolean formula and w is a non-zero integer or infinity. A soft formula is called *positive* if its weight is positive while it is called *negative* if its weight is negative.

Given an EWPM instance ϕ^E and a truth assignment I^E, the *benefit* of I^E on ϕ^E, noted $benefit(\phi^E, I^E)$, is the sum of the weights of the soft formulas satisfied by I^E if I^E satisfies all the hard formulas, otherwise, $benefit(\phi^E, I^E) = -\infty$. ϕ^E is *satisfiable* if ϕ^E has a truth assignment which satisfies all the hard formulas, otherwise it is *unsatisfiable*. The EWPM problem for an EWPM instance ϕ^E is the problem of finding an optimal assignment I^E of ϕ^E that maximizes $benefit(\phi^E, I^E)$, that is, $benefit(\phi^E, I^E) \geq benefit(\phi^E, I^{E\prime})$ for an arbitrary assignment $I^{E\prime}$.

Definition 3.1. (EWPM-to-WPM Transformation).

Let $\phi^E = \{(F_1, w_1), \ldots, (F_m, w_m), (F_{m+1}, \infty), \ldots, (F_{m+m'}, \infty)\}$ be an EWPM instance where w_i is non-zero integer and (F_i, w_i) is a soft formula $(1 \leq i \leq m)$. The EWPM-to-WPM transformation consists of two steps:

(1) For each soft formula (F_i, w_i), a new variable b_i is introduced. The soft formula is then transformed into a soft clause:

(i) (b_i, w_i) *if* $w_i > 0$
$(\neg b_i, -w_i)$ *if* $w_i < 0$

and two hard formulas ensuring that F_i and b_i are logically equivalent:

(ii) $(F_i \rightarrow b_i, \infty)$

(iii) $(b_i \rightarrow F_i, \infty)$

(2) Transform hard formulas into hard clauses with a satisfiability preserving CNF transformation. In this step, any satisfiability preserving CNF transformations could be applied [12].

The two hard formulas ensure that F_i and b_i are logically equivalent.

For a positive soft formula (F_i, w_i), if there is a clause C_i which is satisfiability-equivalent to F_i, then b_i is unnecessary. We simply transform such (F_i, w_i) into (C_i, w_i). Similarly for a negative soft formula (F_i, w_i), if there is a clause C_i which is satisfiability-equivalent to $\neg F_i$, we simply transform such (F_i, w_i) into $(C_i, -w_i)$.

Lemma 3.2. *Let ϕ^E be a satisfiable EWPM instance, ϕ be a WPM instance obtained from ϕ^E by applying EWPM-to-WPM transformation, and W_{neg} be the sum of all negative weights in ϕ^E.*

If I^E is a truth assignment of ϕ^E and satisfies all the hard formulas in ϕ^E, then there exists a truth assignment I of ϕ such that I satisfies all the hard clauses in ϕ and $benefit(\phi^E, I^E) = benefit(\phi, I) + W_{neg}$.

Proof. We assume that I^E satisfies p negative soft formulas and falsifies q negative soft formulas. That is, I^E satisfies (F_i^s, w_i^s) and falsifies (F_j^f, w_j^f) where $w_i^s < 0$ and $w_j^f < 0$ $(1 \leq i \leq p, 1 \leq j \leq q)$. Then, $benefit(\phi^E, I^E) = W_{pos}^{I^E} + \sum_{i=1}^{p} w_i^s$ where $W_{pos}^{I^E}$ denotes the sum of the weights of positive soft clauses satisfied with I^E. We should notice that $\sum_{i=1}^{p} w_i^s + \sum_{j=1}^{q} w_j^f = W_{neg}$.

Now, we make a truth assignment I of ϕ by extending I^E so as to satisfy all the hard clauses in ϕ as follows. If I^E satisfies a soft clause (F_i, w_i), we let I satisfy b_i which is a new variable introduced by the transformation, otherwise, we let I falsify b_i. It is obvious that I satisfies the hard formulas (ii) and (iii) in the transformation. Thus, we can make I satisfy all the hard clauses in ϕ because we use a satisfiability-preserving CNF transformation in (2) of the transformation. From p negative soft formulas (F_i^s, w_i^s), p soft clauses $(\neg b_i^s, -w_i^s)$ are obtained, and, from q negative soft formulas (F_j^f, w_j^f), q soft clauses $(\neg b_j^f, -w_j^f)$ are obtained by the transformation where b_i^s and b_j^f are new variables introduced. I falsifies p soft clauses $(\neg b_i^s, -w_i^s)$ and satisfies q soft clauses $(\neg b_j^f, -w_j^f)$. Thus, $benefit(\phi, I) = W_{pos}^{I} + \sum_{j=1}^{q}(-w_j^f)$ where W_{pos}^{I} denotes the sum of the weights of positive soft clauses satisfied with I. Here, $W_{pos}^{I} = W_{pos}^{I^E}$ because the transformation does not change the satisfiability of positive soft formulas. Therefore, $benefit(\phi, I) = W_{pos}^{I^E} + \sum_{j=1}^{q}(-w_j^f)$.

The difference between $benefit(\phi^E, I^E)$ and $benefit(\phi, I)$ is $benefit(\phi^E, I^E) - benefit(\phi, I) = \sum_{i=1}^{p} w_i^s - \sum_{j=1}^{q}(-w_j^f) = \sum_{i=1}^{p} w_i^s + \sum_{j=1}^{q} w_j^f = W_{neg}$. Consequently, $benefit(\phi^E, I^E) = benefit(\phi, I) + W_{neg}$. □

Lemma 3.3. *Let ϕ^E, ϕ, and W_{neg} be the same as those in Lemma 3.2. If I is a truth assignment of ϕ and satisfies all the hard clauses in ϕ, then there exists a truth assignment I^E of ϕ^E such that I^E satisfies all the hard formulas in ϕ^E and $benefit(\phi^E, I^E) = benefit(\phi, I) + W_{neg}$.*

Proof. We make a truth assignment I^E by restricting I to variables in ϕ^E. By a similar way in the proof of Lemma 3.2, we conclude I^E satisfies the above conditions. □

By Lemma 3.2 and 3.3, we conclude the following theorem.

Theorem 3.4. *Let ϕ^E be a satisfiable EWPM instance and ϕ be a WPM instance obtained from ϕ^E by applying EWPM-to-WPM transformation. Then, ϕ is satisfiable. Furthermore, if I^E_{sol} is an EWPM solution of ϕ^E and I_{sol} is a WPM solution of ϕ, then $\mathit{benefit}(\phi^E, I^E_{\text{sol}}) = \mathit{benefit}(\phi, I_{\text{sol}}) + W_{neg}$ where W_{neg} is the sum of all negative weights in ϕ^E.*

Proof. According to Lemma 3.2, there exists a truth assignment I' of ϕ such that $\mathit{benefit}(\phi^E, I^E_{\text{sol}}) = \mathit{benefit}(\phi, I') + W_{neg}$. According to Lemma 3.3, there exists a truth assignment I'^E of ϕ^E such that $\mathit{benefit}(\phi^E, I'^E) = \mathit{benefit}(\phi, I_{\text{sol}}) + W_{neg}$.

Here, $\mathit{benefit}(\phi^E, I^E_{\text{sol}}) \geq \mathit{benefit}(\phi^E, I'^E)$ and $\mathit{benefit}(\phi, I_{\text{sol}}) \geq \mathit{benefit}(\phi, I')$ because $\mathit{benefit}(\phi^E, I^E_{\text{sol}})$ and $\mathit{benefit}(\phi, I_{\text{sol}})$ are maximal benefits. These inequalities and the above two equalities imply $\mathit{benefit}(\phi, I_{\text{sol}}) = \mathit{benefit}(\phi, I')$ and $\mathit{benefit}(\phi^E, I^E_{\text{sol}}) = \mathit{benefit}(\phi, I_{\text{sol}}) + W_{neg}$. □

3.2.2 Redundancy in Transformation

The hard formula (ii) for a positive soft formula and the hard formula (iii) for a negative soft formula are redundant for solving EWPM problem. That is, without these hard formulas, Theorem 3.4 holds.

Proposition 3.5. *Let ϕ^E be a satisfiable EWPM instance and ϕ be a WPM instance obtained from ϕ^E by applying EWPM-to-WPM transformation. Let ϕ^- denote a multiset of weighted clauses which are soft clauses in ϕ, hard clauses in ϕ from hard formulas in ϕ^E, (ii) for negative soft formulas or (iii) for positive soft formulas. In other words, ϕ^- is obtained from ϕ by eliminating the hard clauses from (ii) for positive soft formulas and (iii) for negative soft formulas.*

If I_{sol} is a WPM solution of ϕ and I^-_{sol} is a WPM solution of ϕ^-, then $\mathit{benefit}(\phi, I_{\text{sol}}) = \mathit{benefit}(\phi^-, I^-_{\text{sol}})$.

Proof. ϕ is a superset of ϕ^-, and the soft clauses in ϕ and those in ϕ^- are the same, so I_{sol} is a truth assignment of ϕ^- and satisfies all the hard clauses in ϕ^-. Therefore, $\mathit{benefit}(\phi, I_{\text{sol}}) \leq \mathit{benefit}(\phi^-, I^-_{\text{sol}})$ because I^-_{sol} gives the maximal benefit of ϕ^-.

Next, we prove $\mathit{benefit}(\phi, I_{\text{sol}}) \geq \mathit{benefit}(\phi^-, I^-_{\text{sol}})$ by showing that I^-_{sol} satisfies all the hard clauses in ϕ.

Consider a positive soft formula $(F_i, w_i)(w_i > 0)$. For this soft formula, the soft clause (b_i, w_i) is generated. Assume that I^-_{sol} falsifies the hard formula (ii) $(F_i \to b_i, \infty)$, then I^-_{sol} satisfies F_i and falsifies b_i. A truth assignment, that agrees to I^-_{sol} except only for the assignment to b_i, gives a benefit of ϕ^-, which is w_i greater than $\mathit{benefit}(\phi^-, I^-_{\mathrm{sol}})$. This contradicts the maximality of $\mathit{benefit}(\phi^-, I^-_{\mathrm{sol}})$. Consequently, I^-_{sol} has to satisfy the hard formula (ii). Thus, I^-_{sol} satisfies the hard clauses from (ii) while these clauses are not contained in ϕ^-.

By the similar way, we can show that I^-_{sol} satisfies the hard clauses obtained from the hard formula (iii) for a negative soft formula. Consequently, I^-_{sol} satisfies all the hard clauses in ϕ. □

3.2.3 Considerations

A natural interpretation of a soft formula (F, w) is that we gain w dollars when F is true. Under the interpretation, a negative soft formula $(F, -20)$ means that we lose 20 dollars when F is true. $(F, -20)$ is transformed into $(\neg b, 20)$ where b is logically equivalent to F. $(\neg b, 20)$ means that we gain 20 dollars when b, i.e. F is false. Someone may consider this is strange because the original soft formula $(F, -20)$ says nothing when F is false. It is ordinary for us to imply that neither gain nor loss when F is false according to the soft formula.

This strange meaning is resolved by the following settings: We pay a deposit of $-w$ dollars for a negative soft formula $(F, w)(w < 0)$ and have $-w$ dollars refunded when F is false.

Take $(F, -20)$ as an example. We first pay 20 dollars as the deposit, then examine the truth value of F. When F is false, we have 20 dollars refunded. Thus, the final payoff is 0 when F is false. This is in accordance with $(F, -20)$. When F is true, 20 dollars should be paid according to $(F, -20)$. Since the deposit 20 dollars has been paid beforehand, this amount of money is taken into the consideration.

We should notice that the total deposit is $-W_{\mathrm{neg}}$ dollars under the assumption of Theorem 3.4. Thus, the expression $(\mathit{benefit}(\phi^E, I^E_{\mathrm{sol}}) = \mathit{benefit}(\phi, I_{\mathrm{sol}}) + W_{neg}\ )$ in Theorem 3.4 means that our maximal gain from ϕ^E equals the difference between maximal gain from ϕ and the total deposit.

Let us leave the interpretation and turn to MinSAT which is an alternative to MaxSAT. MinSAT is the problem of finding a truth assignment that satisfies all the hard clauses

and minimizes the sum of weights of satisfied soft clause. We can say that EWPM "minimizes" the sum of absolute values of weights of satisfied negative soft formulas. From this point of view, we can solve MinSAT with EWPM.

Let $\phi^{\mathrm{Min}} = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ be a satisfiable weighted partial MinSAT instance. We make an EWPM instance ϕ^{Max} by negating the weights in ϕ^{Min}:

$$\phi^{\mathrm{Max}} = \{(C_1, -w_1), \ldots, (C_m, -w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$$

If a truth assignment satisfies all the hard clauses in ϕ^{Min}, then it satisfies all the hard clauses in ϕ^{Max}, and vice versa. It must be noted that $\mathit{benefit}(\phi^{Max}, I) = -\mathit{benefit}(\phi^{Min}, I)$ where I is a truth assignment. This implies that $\mathit{benefit}(\phi^{Max}, I)$ becomes larger as $\mathit{benefit}(\phi^{Min}, I)$ becomes smaller. Consequently, a MinSAT solution of ϕ^{Min}, which gives the minimum benefit of ϕ^{Min}, is a MaxSAT solution of ϕ^{Max}, which gives the maximum benefit of ϕ^{Max}. Reversely, a MaxSAT solution of ϕ^{Max} is a MinSAT solution of ϕ^{Min}.

In short, ϕ^{Min} and ϕ^{Max} have the same solution; accordingly, we may say that EWPM is not only an extension of MaxSAT but also that of MinSAT, or EWPM is an integration of MaxSAT and MinSAT.

3.3 Chapter Summary

This chapter presented an extension of WPM to deal with not only positive weights but also negative weights. Specifically, in order to solve EWPM instances with WPM solvers, we gave a transformation from EWPM to WPM and examined the relationship between their solutions. We theoretically proved that, for a WPM instance and its extension, the optimal solution of EWPM is always $(-W_{neg})$ larger than that of WPM, where W_{neg} is the sum of negative weights in the EWPM instance. The extended WPM paves a way for a wider range of applications, such as the coalition structure generation problem described in the subsequent chapters.

Chapter 4

MaxSAT Encoding for the CSG Problem based on Rule Relations

Coalition Structure Generation (CSG) is a main research issue in the domain of coalition games. A majority of existing works assume that the value of a coalition is independent of others in the coalition structure. Recently, there has been interest in more realistic settings, where the value of a coalition is affected by the formation of other coalitions. This effect is known as *externality*.

The focus of this chapter is to make use of weighted partial MaxSAT to solve the CSG problem where externalities may exist. Motivated by the previous works that represent the CSG problem in a set of rules, we encode rule relations and their constraints into weighted partial MaxSAT formulas and show that MaxSAT solvers are effective in solving the CSG problem with and without externalities. Specifically, Section 4.1 introduces the CSG problem and its concise representation schemes used in this chapter. Section 4.2 introduces related works on the CSG problem, including an overview of the previous works and the direct encoding algorithm that is the foundation of the next chapter. Section 4.3 describes the rule relation-based WPM approach encoding the previous algorithm into weighted partial MaxSAT. The evaluation results are shown in Section 4.4 and conclusion is given in Section 4.5.

4.1 Coalition Structure Generation (CSG)

Coalition formation, the process by which agents come together and take joint actions to perform a set of tasks cooperatively, is an important capacity in multi-agent systems. Sample applications of coalition formation include distributed vehicle routing [98], sensor network [23], and e-commerce [110]. Coalitional games have been proved highly influential in the research of multi-agent systems as they model the ability of the agents to act as a coherent group as primitives [76]. Coalition Structure Generation (CSG), one of the main challenges in coalition formation, is the main research issue in the use of coalitional games in multi-agent systems [97]. The CSG problem involves partitioning a set of agents so that the total value of all coalitions is maximized.

A majority of the existing works assume that a coalition's value is independent of other coalitions in the coalition structure. Such settings are known as Characteristic Function Game (CFGs), where the value of a coalition is given by a *characteristic function*. Many, but clearly not all, real-world multi-agent problems happen to be CFGs [97, 98]. Recently, there has been interest in a more realistic *partition function* form of coalition values, where the value of a coalition is affected by the formation of other coalitions. This class of coalitional games is called Partition Function Games (PFGs). The effect is known as *externality*. Examples of games with externalities include collusion in oligopolies, congestion games, as well as various forms of international policy coordination between countries [14, 84].

Given a set of agents A, a coalition, denoted by C, is a non-empty subset of A, i.e., $C \in 2^A \backslash \emptyset$. A *coalition structure*, CS, is an exhaustive set of mutually disjoint coalitions over A, i.e., CS is subject to the constraints: $\forall i, j \left(i \neq j\right), C_i \cap C_j = \emptyset$, $\bigcup_{C_i \in CS} C_i = A$. We denote by $\Pi\left(A\right)$ the set of all coalition structures over A.

4.1.1 Characteristic Function Game

In a setting without externalities, the value of a coalition C is given by a *characteristic function* $v : 2^A \rightarrow \mathbb{R}$, assigning a real-valued payoff to each coalition $C \subseteq A$. The value of a coalition structure CS is called *social welfare*, denoted as $V\left(CS\right)$, given by $V\left(CS\right) = \sum_{C_i \in CS} v\left(C_i\right)$. The objective of solving the CSG problem is to find an optimal coalition structure that makes the social welfare maximized, i.e., given A, find CS^* such that $\forall CS \in \Pi\left(A\right), V\left(CS^*\right) \geq V\left(CS\right)$.

Example 4.1. *We reconsider the game in Example 1.1. For* $A = \{a_1, a_2, a_3\}$*, the characteristic function is denoted as:*

$$v(\{a_1\}) = 1, \quad v(\{a_2\}) = 0, \quad v(\{a_3\}) = 0,$$
$$v(\{a_1, a_2\}) = 1, \quad v(\{a_1, a_3\}) = 1, \quad v(\{a_2, a_3\}) = 1,$$
$$v(\{a_1, a_2, a_3\}) = 2.$$

There are 5 possible coalition structures, the social welfare of all these coalition structures are enumerated as follows.

$$V(\{\{a_1\}, \{a_2\}, \{a_3\}\}) = 1, \quad V(\{\{a_1, a_2\}, \{a_3\}\}) = 1,$$
$$V(\{\{a_1, a_3\}, \{a_2\}\}) = 1, \quad V(\{\{a_1\}, \{a_2, a_3\}\}) = 2,$$
$$V(\{\{a_1, a_2, a_3\}\}) = 2.$$

It is clear that there are two optimal coalition structures: $\{\{a_1\}, \{a_2, a_3\}\}$ *and* $\{\{a_1, a_2, a_3\}\}$*, with the social welfare of 2.*

Naive representation of characteristic functions enumerates the payoffs to each possible set of agents, requiring space exponential in the number of agents. To be more specific, the representation size the CSG problem is $\mathrm{O}(2^n)$ for CFGs, where n is the number of agents. This is prohibitive for large n. Ieong and Shoham [43] develop a concise representation called *marginal contribution network (MC-net)*, which largely reduces the space necessary for representation.

Definition 4.1. (MC-nets). An MC-net consists of a set of *rules* R. Each rule $r_i \in R$ is expressed in a syntactic form $I_i \to w_i$, where $w_i \in \mathbb{R}$ and I_i is the condition of rule r_i, denoted by a conjunction of literals over A, i.e., $\{a_1 \wedge a_2 \wedge \cdots \wedge a_l \wedge \neg a_{l+1} \wedge \cdots \wedge \neg a_m\}$. For each rule r_i, we call P_i *positive literals* where $P_i = \{a_j\}_{j=1}^{l}$, and N_i *negative literals* where $N_i = \{a_j\}_{j=l+1}^{m}$. A rule r_i is said to *apply to* a coalition C if $P_i \subseteq C$ and $N_i \cap C = \emptyset$, i.e., all agents in P_i are in C, and none of agents in N_i are in C. For a coalition C, $v(C) = \sum_{r_i \in R'} w_i$, where R' is the set of rules that apply to C. Thus, the value of a coalition structure CS is given as $\sum_{C \in CS} v(C)$. Without loss of generality, we assume each rule has at least one positive literal.

Thus, the problem of finding CS^* is equivalent to finding a set of rules that apply to some coalition $C \in CS^*$, such that the sum of values in the set of rules is maximized.

In practice, the procedure for checking whether a rule $r\,(:I_r \to w_r)$ applies to a coalition C can proceed in the following way. First assign *true* to all the agents present in C and $false$ to others. Based on such assignment, examine the truth value of I_r. If I_r evaluates to true, the rule r applies to C and the value of w_r is obtained. Otherwise r does not apply to C and the corresponding gain is zero.

Example 4.2. *The game in Example 4.1 can be represented with just two rules:*

- $r_1 : a_1 \to 1$,
- $r_2 : a_2 \wedge a_3 \to 1$.

r_1 applies to coalition $\{a_1\}$, $\{a_1, a_2\}$, $\{a_1, a_3\}$, and $\{a_1, a_2, a_3\}$. r_2 applies to $\{a_2, a_3\}$ and $\{a_1, a_2, a_3\}$. Both rules apply to $\big\{\{a_1\}, \{a_2, a_3\}\big\}$ and $\big\{\{a_1, a_2, a_3\}\big\}$, thus there are two optimal coalition structures: $\big\{\{a_1\}, \{a_2, a_3\}\big\}$ and $\big\{\{a_1, a_2, a_3\}\big\}$. The maximized social welfare is $V\,(CS^*) = 2$.

4.1.2 Partition Function Game

In a setting with externalities, an *embedded coalition* is a pair (C, CS). Let M denote the set of all embedded coalitions, i.e., $M := \{(C, CS) : CS \in \Pi\,(A)\,, C \in CS\}$. The value of a coalition depends on the formation of other co-existing coalitions in the coalition structure, and is specified by a *partition function*, which is a mapping $w : M \to \mathbb{R}$. A game in a partition function form is a tuple (A, w). The problem of solving CSG seeks a coalition structure CS, such that $V\,(CS^*) = \sum_{C \in CS^*} w\,(C, CS^*)$ is optimized.

Example 4.3. *We reconsider the game in Example 1.2. For $A = \{a_1, a_2, a_3\}$, the partition function is denoted as:*

$$
\begin{aligned}
&w\big(\{a_1\}, \big\{\{a_1\}, \{a_2\}, \{a_3\}\big\}\big) = 1, \quad && w\big(\{a_1\}, \big\{\{a_1\}, \{a_2, a_3\}\big\}\big) = 1.5,\\
&w\big(\{a_2\}, \big\{\{a_1\}, \{a_2\}, \{a_3\}\big\}\big) = 0, \quad && w\big(\{a_2\}, \big\{\{a_2\}, \{a_1, a_3\}\big\}\big) = 0,\\
&w\big(\{a_3\}, \big\{\{a_1\}, \{a_2\}, \{a_3\}\big\}\big) = 0, \quad && w\big(\{a_3\}, \big\{\{a_3\}, \{a_1, a_2\}\big\}\big) = 0,\\
&w\big(\{a_1, a_2\}, \big\{\{a_1, a_2\}, \{a_3\}\big\}\big) = 1, \quad && w\big(\{a_1, a_3\}, \big\{\{a_2\}, \{a_1, a_3\}\big\}\big) = 1,\\
&w\big(\{a_2, a_3\}, \big\{\{a_1\}, \{a_2, a_3\}\big\}\big) = 1, \quad && w\big(\{a_1, a_2, a_3\}, \big\{\{a_1, a_2, a_3\}\big\}\big) = 2.
\end{aligned}
$$

There are 5 possible coalition structures, the social welfare of these coalition structures are listed as follows.

$$V\left(\{\{a_1\},\{a_2\},\{a_3\}\}\right)=1, \qquad V\left(\{\{a_1,a_2\},\{a_3\}\}\right)=1,$$
$$V\left(\{\{a_1,a_3\},\{a_2\}\}\right)=1, \qquad V\left(\{\{a_1\},\{a_2,a_3\}\}\right)=2.5,$$
$$V\left(\{\{a_1,a_2,a_3\}\}\right)=2.$$

By enumerating all possible coalition structures and comparing their social welfare, we find that the optimal coalition structure is $\{\{a_1\},\{a_2,a_3\}\}$*, with the social welfare of 2.5.*

Coalitional games with externalities are more complex since the value depends not only on individual coalitions, but also on embedded coalitions. The representation size increases to $\mathrm{O}(n^n)$ for PFGs [87]. Therefore, solving the CSG problem for PFGs is more challenging than the CFG case, which is already NP-complete [97]. Michalak et al. [69] develop a concise representation scheme for PFGs by extending MC-nets to a partition function, called *embedded MC-nets*.

Definition 4.2. (Embedded MC-nets). An embedded MC-net is given by a set of *embedded rules* ER. Each rule $er \in ER$ is expressed in a syntactic form $I_0|I_1,\cdots,I_k \rightarrow w_{er}$, where each $I_i : i \in \{0,\cdots,k\}$ is conjunction of literals over A. A rule er is said to *apply to* an embedded coalition (C,CS) if

1. I_0 applies to C, and
2. each $I_i : i \in \{1,\cdots,k\}$ applies to at least one coalition in $CS\backslash C$.

For an embedded coalition (C,CS), $w(C,CS)=\sum_{er\in ER'} w_{er}$, where ER' is the set of embedded rules that apply to (C,CS).

Example 4.4. *The game with externalities in Example 4.3 can be described with the following set of rules:*

- $r_1 : a_1 \rightarrow 1$,
- $r_2 : a_2 \wedge a_3 \rightarrow 1$,
- $r_3 : a_1|a_2 \wedge a_3 \rightarrow 0.5$.

r_1 *applies to coalition* $\{a_1\}$, $\{a_1,a_2\}$, $\{a_1,a_3\}$, *and* $\{a_1,a_2,a_3\}$. r_2 *applies to* $\{a_2,a_3\}$ *and* $\{a_1,a_2,a_3\}$. r_3 *applies to an embedded coalition* $\left(\{a_1\},\{\{a_1\},\{a_2,a_3\}\}\right)$. *All rules*

apply to $\{\{a_1\},\{a_2,a_3\}\}$*, thus the optimal coalition structure is* $\{\{a_1\},\{a_2,a_3\}\}$ *and the optimized social welfare is* $V(CS^*) = V(\{\{a_1\},\{a_2,a_3\}\}) = 2.5$.

It worth noting that in this example, $\{a_1,a_2,a_3\}$ *is no longer the optimal solution because in this case,* r_3 *does not apply to the grand coalition and the corresponding payoff of 0.5 cannot be obtained.*

In Example 4.4, r_1 and r_3 highlight the difference between games with and without externalities. In r_1, agent a_1 contributes to any embedded coalition it belongs with value 1. Additionally, r_3 said that if there co-exists another embedded coalition where a_2 and a_3 cooperate, the value of a_1 is increased by 0.5. This happens in $\{\{a_1\},\{a_2,a_3\}\}$ and $\{\{a_1\},\{a_2,a_3\}\}$. In this way, embedded MC-nets allow to capture externalities in coalitional games [69].

4.2 Related Works

4.2.1 An Overview

There have been a number of attempts to solve the CSG problem for CFGs, broadly classified as dynamic programming and anytime optimal algorithms. Dynamic programming (DP) based algorithms [86, 93] break the optimization problem into sub-problems that can be solved recursively, and then combine the results of the sub-problems to output a final solution. These algorithms have the lowest worst complexity, i.e., given n agents, they guarantee to find an optimal solution in $\Theta(3^n)$ time. However, they cannot generate a solution until they complete the entire execution. By contrast, anytime optimal algorithms, e.g., integer partition (IP) [85, 90], can return a solution even if they are terminated prematurely. The solution quality improves monotonically as the computation time increases. Nevertheless, these algorithms provide worst-case guarantees on the quality of the optimal solution in $\Theta(n^n)$, i.e., in the worst case, the algorithms need to search the entire space. In practice, IP has been shown to significantly outperform DP based algorithms for many popular test distributions of coalition values [89]. Recent algorithms [89, 101] combine these two techniques, so as to overcome the limitations of DP and IP. Another branch of algorithms is called Heuristics [100, 103]. Although they return solutions more quickly than the other two techniques, they provide no guarantee on finding the optimal. In the worst case, they might never find the optimal solution.

To date, few works have attempted to solve the CSG problem in PFGs. The initial work is under the restrictions that the externalities are either always positive or always negative [68, 87]. These constraints were relaxed in [8], which incorporates mixed externalities in the agent types-based framework, i.e., positive and negative externalities could co-exist in a problem instance. These works represent partition functions by listing the values of all possible coalitions, thus requiring space exponential in the number of agents in the game [43]. When the number of agents n becomes large, solving the problem with the naive representation is prohibitive. What is why these works could only solve problem instances with only a few dozen of agents.

Another line of research focuses on solving the CSG problem with concise representations [75, 88, 111, 112], where a characteristic function or a partition function is represented by a set of rules and the representation size can be reduced significantly. In this context, Ohta et al., [75] initially formalize the CSG problem as a problem of finding the subset of rules that maximizes the sum of rule values under certain constraints. Their work takes an initial step towards developing efficient constraint optimization algorithms for solving the CSG problem in CFGs, and shows that with concise representations such as MC-net, the CSG problem could be solved within significantly less time than other works. In case of PFGs, applying compact representation schemes to the CSG problem is more desirable, as PFGs require greater space than CFGs if we use the naive representation. In this context, embedded MC-net [69], a compact representation designed for coalitional games with externalities, has been developed. By using the embedded MC-net, Ueda et al. [111] extend the formalization of CSG in [75] and develop a direct encoding algorithm to handle the externalities among the CSG problem.

4.2.2 Direct Encoding

In MC-nets or embedded MC-nets, each (embedded) rule is associated with a real-valued payoff. If the value is positive, the rule is called *positive value (embedded) rule*. If the value is negative, the rule is called *negative value (embedded) rule*. This section exhibits the existing works on handling positive and negative value (embedded) rules, respectively, which are the most related to our WPM encodings introduced in the next section.

Definition 4.3. [75]. (Feasible rule set). A set of rules R is *feasible* if there exists a coalition structure CS where each rule in R applies to some coalition C or embedded coalition (C, CS), where $C \in CS$.

Thereby, the problem of finding CS^* is equivalent to finding a feasible rule set, such that the sum of values of the rule set is maximized.

To handle positive value rules in MC-nets, Ohta et al. [75] represent an MC-net in a rule relations-based graph. In this graph, each vertex is a rule, and between any two vertices, there exists an edge whose type is one of the four cases described as follows.

Definition 4.4. [75]. (Relations between rules). The possible relations between rules r_i and r_j are classified into the following four non-overlapping and exhaustive cases:

- Compatible on the same coalition: $P_i \cap P_j \neq \emptyset$ and $P_i \cap N_j = P_j \cap N_i = \emptyset$.
- Compatible on different coalitions: $P_i \cap P_j = \emptyset$ and ($P_i \cap N_j \neq \emptyset$ or $P_j \cap N_i \neq \emptyset$).
- Incompatible: $P_i \cap P_j \neq \emptyset$ and ($P_i \cap N_j \neq \emptyset$ or $P_j \cap N_i \neq \emptyset$).
- Independent: $P_i \cap P_j = \emptyset$ and $P_i \cap N_j = P_j \cap N_i = \emptyset$.

Example 4.5. *[75] Let there be five agents a, b, c, d, e and four rules:*

$$r_1 : b \wedge e \rightarrow 3, \qquad r_2 : a \wedge b \wedge c \wedge \neg d \rightarrow 2,$$
$$r_3 : a \wedge d \rightarrow 1, \qquad r_4 : c \wedge e \rightarrow 1.$$

The relations among these rules are listed as follows:

- *r_1 and r_2 are compatible on the same coalition.*
- *r_1 and r_4 are compatible on different coalitions.*
- *r_2 and r_3 are incompatible.*
- *r_2 and r_4 are compatible on the same coalition.*
- *Other unmentioned rule relations are independent.*

The graph representation of Example 4.5 is depicted in Fig. 4.1 [75] where the condition of each rule is denoted in the form of (P_i, N_i). For instance, $(\{a, b, c\}, \{d\})$ represents the condition of r_1, i.e., $a \wedge b \wedge c \wedge \neg d$.

The following constraints are proved to be sufficient to find out a feasible rule set among positive value rules in MC-nets. We refer readers to [75] for the proof of the theorem.

Theorem 4.5. *[75]. A set of rules R' is feasible if and only if R' satisfies the following conditions.*

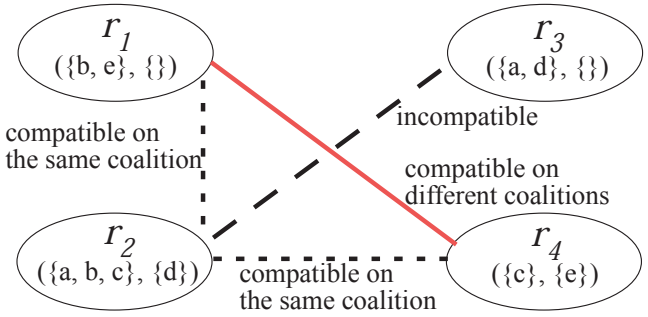


FIGURE 4.1: Graphical representation of Example 4.5

1. *R' includes no pair of rules/vertices connected by an "incompatible" edge, and*
2. *if two rules in R' have the relation of "compatible on different coalitions", then they are not reachable via "compatible on the same coalition" edges in R' .*

To handle positive value embedded rules, Ueda et al. [111] develop an *explicit form* for each embedded rule $er \in ER$ (i.e., $er : I_0|I_1, \cdots, I_k \rightarrow w_{er}$) by adding each positive literal in I_0 to the negative literals of each $I_1, \cdots, I_k$, as well as adding each positive literal in $I_1, \cdots, I_k$ to the negative literals of I_0. Thus, er applies to some embedded coalition in CS if each of $I_0, \cdots, I_k$ applies to some coalition in CS. The explicit form relieves the algorithm of checking whether the coalitions I_0 and I_i $(i = 1, \ldots, k)$ apply to are different.

When all (embedded) rules have positive values, the solver in [75] only needs to select as many rules as possible because satisfying more rules never decreases the social welfare. However, when negative values are involved, extra constraints are required to specify when negative value (embedded) rules should be selected, otherwise, the solver would simply ignore these rules because selecting them decreases the social welfare. This would lead to incorrect results since in some cases, choosing some negative value (embedded) rules is the precondition of satisfying more positive value (embedded) rules.

To handle negative value (embedded) rules, Ueda et al. [111] create zero-valued dummy rules for each negative value (embedded) rule, and specify their constraint in Theorem 4.6. Readers may refer to the proof in literature [111].

Theorem 4.6. *[111]. A negative value (embedded) rule applies to a coalition in coalition structure CS if and only if none of its dummy rules apply to any coalition in CS.*

4.3 WPM Encoding on Rule Relations

In this section, we set out to encode rule relations and the technique of direct encoding into WPM. First, we consider a special case that a rule contains only one agent. In this case, the rule is always selected because no matter how a CS is structured, a rule with a single agent a always applies to a coalition that contains a. Without loss of generality, in the rest of this chapter, we assume each rule contains at least two agents, one of which must be positive literal.

4.3.1 Encoding Positive Value Rules

Encoding a positive value rule into WPM formulas is straightforward. Let $R = \{r_1, \ldots, r_n\}$ be a set of positive value rules. For each rule r_i, we introduce a new Boolean variable B_i $(i = 1, \ldots, n)$. We define $\mathcal{B}$ as the set of all such Boolean variables, i.e., $\mathcal{B} = \{B_1, \ldots, B_n\}$. Intuitively, $B_i = 1$ means r_i is in a feasible set of rules.

In order to deal with the reachability mentioned in Theorem 4.5, we introduce a Boolean variable $S_{i,j}$ for each pair of rules r_i and r_j where $1 \leq i < j \leq n$. Intuitively, $S_{i,j} = 1$ means both r_i and r_j are reachable via "compatible on the same coalition" edges in the feasible rule set. To accomplish the reachability, we need the following hard clauses:

- $\neg S_{i,j} \vee \neg S_{j,k} \vee S_{i,k}$,
- $\neg S_{i,j} \vee \neg S_{i,k} \vee S_{j,k}$,
- $\neg S_{i,k} \vee \neg S_{j,k} \vee S_{i,j}$,

where $1 \leq i < j < k \leq n$. The number of hard clauses for representing the transitive laws over R is $n \cdot (n-1) \cdot (n-2)/2$.

Definition 4.7. (WPM encoding of positive value rules). Let R be a set of positive value rules. For each positive value rule $r_i \in R$ (i.e., $r_i : I_i \to w_i$, $w_i > 0$), we introduce a weighted soft clause (B_i, w_i). The possible relations between two rules r_i and r_j are encoded as following hard clauses:

(1) If r_i and r_j are "compatible on the same coalition", three hard clauses are generated: $\neg B_i \vee \neg B_j \vee S_{i,j}$, $\neg S_{i,j} \vee B_i$, and $\neg S_{i,j} \vee B_j$.

(2) If r_i and r_j are "compatible on different coalitions ", one hard clause is generated: $\neg B_i \vee \neg B_j \vee \neg S_{i,j}$.

(3) If r_i and r_j are "incompatible", one hard clause is generated: $\neg B_i \vee \neg B_j$.

(4) If r_i and r_j are " independent ", then no clause is generated.

Definition 4.7 encodes the rule relations introduced in Definition 4.4 into WPM formulas. Soft clauses are interpretations of positive value rules. Hard clauses are constraints on rules, which characterize whether a rule applies to some coalition in CS.

In the following, we use $Hard(R)$ to denote a set of hard clauses representing the transitive laws and those introduced by Definition 4.7.

Lemma 4.8. *Let R be a set of positive value rules, $R' = \{r'_1, r'_2, \ldots, r'_l\} \subseteq R$ be a feasible rule set, and $\mathcal{B}' = \{B'_1, B'_2, \ldots, B'_l\}$ be the corresponding Boolean variable set. Then, $\mathcal{B}' \cup Hard(R)$ is satisfiable.*

Proof. First, we consider hard clauses introduced by Definition 4.7 (1). If $B_i \in \mathcal{B}'$ and $B_j \in \mathcal{B}'$, then we can make $S_{i,j} = 1$ because $r_i \in R'$, $r_j \in R'$, and r_i and r_j are compatible on the same coalition. If $B_i \notin \mathcal{B}'$, then we can make $S_{i,j} = 0$ because $r_i \notin R'$. Similarly, if $B_j \notin \mathcal{B}'$, we can make $S_{i,j} = 0$. Thus, hard clauses introduced by Definition 4.7 (1) are satisfied.

Next, we consider hard clauses introduced by Definition 4.7 (2). If $B_i \notin \mathcal{B}'$ or if $B_j \notin \mathcal{B}'$, the truth value of $S_{i,j}$ is not limited, so we only consider the case of both $B_i \in \mathcal{B}'$ and $B_j \in \mathcal{B}'$. In this case, we can make $S_{i,j} = 0$ because r_i and r_j are not reachable according to Theorem 4.5 (2).

Then, we consider hard clauses introduced by Definition 4.7 (3). We assume it is possible to hold both $B_i \in \mathcal{B}'$ and $B_j \in \mathcal{B}'$. According to Theorem 4.5 (1), either $r_i \notin R'$ or $r_j \notin R'$. This contradicts our assumption.

It is obvious that the transitive laws are satisfiable under the above assignment. □

Lemma 4.9. *Let R be a set of positive value rules and $\mathcal{B}' = \{B'_1, B'_2, \ldots, B'_l\}$ be a subset of $\mathcal{B}$. If $\mathcal{B}' \cup Hard(R)$ is satisfiable, then $\{r'_1, r'_2, \ldots, r'_l\}$ is a feasible rule set.*

Proof. The proof is basically the reverse process of proving Lemma 4.8. Given $\mathcal{B}' \cup Hard(R)$ is satisfiable, the corresponding rule set $R' = \{r'_1, r'_2, \ldots, r'_l\}$ includes no pair of rules connected by the relation of "incompatible" because of Definition 4.7-3, and if any two rules in R' are compatible on different coalitions, then they are not reachable

via the relation of "compatible on the same coalition" according to Definition 4.7 (2). Therefore, R' is a feasible rule set. □

The above two lemmas indicate that if R' is a feasible rule set, then $\mathcal{B}' \cup Hard(R)$ is satisfiable, and vice versa. Thus, the following theorem holds.

Theorem 4.10. *The encoding given by Definition 4.7 with the transitive laws leads a MaxSAT solver to output the correct results of the CSG problem, consisting of a set of positive value rules.*

Proof. We need to prove the soft clause encoded in Definition 4.7 is correct, i.e., we show the result output by the MaxSAT solver is equal to the value of the feasible rule set R'.

If r_i is a feasible rule set, then the corresponding B_i evaluates 1, and both of $I_i \to w_i$ and (B_i, w_i) lead to a payoff of w_i, otherwise, $B_i = 0$, and the corresponding payoffs of both formulas are 0. Given a feasible rule set $R' = \{r'_1, r'_2, \ldots, r'_l\}$ and the corresponding Boolean variable set $\mathcal{B}' = \{B'_1, B'_2, \ldots, B'_l\}$, according to Lemma 4.8, $\mathcal{B}' \cup Hard(R)$ is satisfiable. This means $\forall B'_i \in \mathcal{B}'$, (B'_i, w_i) is satisfied, thus the result output by the MaxSAT solver is $\sum_{i=1}^{l} w_i$, which is equal to the value of the feasible rule set R'. As an alternative, we can also prove the sum of weights of satisfied soft clauses is equal to the value of the feasible rule set R' by Lemma 4.9. □

4.3.2 Encoding Positive Value Embedded Rules

For an embedded rule $er \in ER$ (i.e., $er : I_0 | I_1, \cdots, I_k \to w_{er}$) that applies to (C, CS), an implicit constraint is that each $I_1, \cdots, I_k$ must apply to some other coalition except C. To represent the constraint explicitly, Ueda et al. [111] develop an *explicit form* for each embedded rule by adding each positive literal in I_0 to the negative literals of each $I_1, \cdots, I_k$, as well as adding each positive literal in $I_1, \cdots, I_k$ to the negative literals of I_0. After an embedded rule is transformed into explicit form, the number of agents may increase dramatically. To be specific, let $pos_0, \cdots, pos_k$ be the number of positive literals in $I_0, \cdots, I_k$, respectively. In the worst case, the number of agents that need to be added in er is $\left(k \cdot pos_0 + \sum_{x=1}^{k} pos_x\right)$. The large number of agents contained in embedded rules would be prohibitive when the number of embedded rules grows, especially in the approach in use of relations between pairs of agents. Therefore, reducing the number of agents in the explicit form is highly in demand.

We reform the explicit form of $er : I_0|I_1, \ldots, I_k \rightarrow w_{er}$. For each I_i ($i \in \{1, \cdots, k\}$), we first investigate whether $P_0 \cap N_i \neq \emptyset$ or $P_i \cap N_0 \neq \emptyset$ holds. If at least one of the conditions holds, it means I_0 and I_i cannot apply to the same coalition by nature, then nothing would be done. Otherwise, we add one agent in P_i to N_0, so that the coalition that I_i applies to (suppose C) contains at least one agent in N_0, which makes I_0 impossible to apply to C. This explicit form guarantees that I_0 and I_i ($i \in \{1, \cdots, k\}$) cannot apply to the same coalition. With this method, the number of agents to be added in er is largely decreased.

Example 4.6. *For an embedded rule* $er : a_1 \wedge a_2 | a_4 \wedge \neg a_3 \rightarrow 1$*, the explicit form developed in [111] for* er *is* $a_1 \wedge a_2 \wedge \neg a_4 | a_4 \wedge \neg a_3 \wedge \neg a_1 \wedge \neg a_2 \rightarrow 1$*, while in our method, the explicit form is simplified as* $a_1 \wedge a_2 \wedge \neg a_4 | a_4 \wedge \neg a_3 \rightarrow 1$*.*

The explicit form has already indicated that for each I_i ($i = 1, \ldots, k$), it is impossible that I_0 and I_i apply to the same coalition. Therefore, to make er apply to some embedded coalition, we only need to have each of $I_0, I_1, \ldots, I_k$ apply to some coalition in CS, without any other constraints. For simplicity, we assume in the rest of this chapter, each embedded rule is expressed in our defined explicit form.

Given a positive value embedded rule $er : I_0|I_1, \ldots, I_k \rightarrow w_{er}$ ($w_{er} > 0$), we introduce Boolean variables B_{er}, $B^0_{er}, \ldots, B^k_{er}$ for $er, r^0_{er}, r^1_{er}, \ldots, r^k_{er}$, respectively, where each r^i_{er} ($i = 0, \ldots, k$) is an auxiliary rule $r^i_{er} : I_i \rightarrow 0$. Intuitively, $B_{er} = 1$ means er applies to some embedded coalition in CS, and $B^i_{er} = 1$ means r^i_{er} applies to some coalition in CS. Obviously, whether an embedded rule applies to some embedded coalition is uniquely determined by whether the set of its auxiliary rules is feasible. Thus the constraints on rule relations as well as the transitive laws for representing reachability could be specified on auxiliary rules only, instead of embedded rules.

Definition 4.11. (WPM encoding for positive value embedded rules). Let ER be a set of positive value embedded rules in the explicit form. For each positive value embedded rule $er \in ER$ (i.e., $er : I_0|I_1, \ldots, I_k \rightarrow w_{er}$ ($w_{er} > 0$)), the following clauses are introduced:

- one weighted soft clause: (B_{er}, w_{er}), and
- $(k + 1)$ hard clauses: $\neg B_{er} \vee B^i_{er}$, for $i = 0, \ldots, k$.

The possible relations between any two auxiliary rules and the transitive laws among auxiliary rules are defined in the same way as introduced in Section 4.3.1.

Theorem 4.12. *The encoding given by Definition 4.11 leads a MaxSAT solver to output the correct results of the CSG problem, consisting of a set of positive value embedded rules.*

Proof. The encoding of rule relations with transitive laws has been elaborated in Section 4.3.1, we only need to prove the encoding of positive value embedded rule in Definition 4.11 is correct.

If each of $B_{er}^0, \ldots, B_{er}^k$ is true, then the value of B_{er} is not restricted by any hard clauses. In this case, the MaxSAT solver would always prefer $B_{er} = 1$, so that the payoff w_{er} could be gained.

If at least one of $B_{er}^0, \ldots, B_{er}^k$ is false, then the corresponding hard clause could not be satisfied unless $B_{er} = 0$. Thus, the corresponding payoff is 0.

In either case, the MaxSAT solver calculate the correct social welfare. □

Example 4.7. *For $A = \{a_1, a_2, a_3, a_4\}$, assume there are four rules:*

$$r_1 : a_1 \wedge a_2 \to 2, \qquad r_2 : a_3 \wedge \neg a_4 \to 1,$$
$$r_3 : a_4 \wedge \neg a_1 \to 1, \qquad r_4 : a_1 \wedge a_2 \wedge \neg a_4 | a_4 \wedge \neg a_3 \to 1.$$

We introduce four weighted soft clauses $(B_1, 2)$, $(B_2, 1)$, $(B_3, 1)$ and $(B_4, 1)$ for r_1, r_2, r_3 and r_4, respectively. Embedded rule r_4 generates two auxiliary rules: r_5: $a_1 \wedge a_2 \wedge \neg a_4 \to 0$, r_6: $a_4 \wedge \neg a_3 \to 0$. The constraint between r_4 and its auxiliary rules is specified by two hard clauses as defined in Definition 4.11: $\neg B_4 \vee B_5$, $\neg B_4 \vee B_6$. The rule relations are captured by the following clauses:

- *r_1 and r_5 are compatible on the same coalition:*

  $\neg B_1 \vee \neg B_5 \vee S_{1,5}$, $\neg S_{1,5} \vee B_1$, $\neg S_{1,5} \vee B_5$.
- *r_3 and r_6 are compatible on the same coalition:*

  $\neg B_3 \vee \neg B_6 \vee S_{3,6}$, $\neg S_{3,6} \vee B_3$, $\neg S_{3,6} \vee B_6$.
- *r_1 and r_3 are compatible on different coalitions:* $\neg B_1 \vee \neg B_3 \vee \neg S_{1,3}$.
- *r_2 and r_3 are compatible on different coalitions:* $\neg B_2 \vee \neg B_3 \vee \neg S_{2,3}$.
- *r_2 and r_6 are compatible on different coalitions:* $\neg B_2 \vee \neg B_6 \vee \neg S_{2,6}$.
- *r_3 and r_5 are compatible on different coalitions:* $\neg B_3 \vee \neg B_5 \vee \neg S_{3,5}$.
- *r_5 and r_6 are compatible on different coalitions:* $\neg B_5 \vee \neg B_6 \vee \neg S_{5,6}$.

- r_1 *and* r_2, r_1 *and* r_6, r_2 *and* r_5 *are independent: no clause is generated.*

To solve this problem, 16 variables and 47 clauses are generated. The generated clauses include 4 weighted soft clauses representing rules, 2 hard clauses generated by the embedded rule, 11 hard clauses capturing the constraints on rules, and 30 hard clauses representing the transitive laws. All the above clauses are connected into a CNF formula. The MaxSAT solver takes the CNF formula as an input, and outputs the optimized social welfare and the corresponding feasible rule set. For the complete file in WPM input format, readers may refer to Appendix A.

4.3.3 Encoding Negative Value Rules

Encoding negative value rules into WPM formulas is not as straightforward as translating positive value rules. First, in WPM, the weight of a clause must be positive, and MaxSAT solvers can only solve problems with the weight larger than 0. Therefore, turning negative values into positive is indispensible. In addition, the constraints on a negative value rule and its dummy rules [111] should be encoded into propositional Boolean formulas that MaxSAT solvers can deal with.

Definition 4.13. (WPM encoding of negative value rules). Let R be a set of negative value rules. We introduce a weighted soft clause $(\neg B_x, -w_x)$ for each negative value rule $r_x \in R$ (i.e., $r_x : I_x \rightarrow w_x$, $w_x < 0$). Suppose $I_x = \{a_1 \wedge a_2 \wedge \cdots \wedge a_k \wedge \neg a_{k+1} \wedge \neg a_{k+2} \wedge \cdots \wedge \neg a_m\}$. r_x generates $m-1$ dummy rules, following two types:

(i) $r_x^i : a_1 \wedge \neg a_{i+1} \rightarrow 0$ where $1 \leq i \leq k-1$,
(ii) $r_x^j : a_1 \wedge a_{j+1} \rightarrow 0$ where $k \leq j \leq m-1$,

where r_x^i denotes the i-th dummy rule created by r_x.

The constraints on r_x and its dummy rules are specified by the following hard clauses:

(1) $B_x \vee \bigvee_{i \in \{1,\ldots,m-1\}} B_x^i$, and
(2) $\neg B_x^i \vee \neg B_x$, for $i = 1, \ldots, m-1$,

where B_x^i is a Boolean variable.

Intuitively, $B_x^i = 1$ if r_x^i applies to a coalition C in CS and $B_x^i = 0$, otherwise. The final social welfare after encoding is $(-W_{neg})$ larger than the original one, where W_{neg} is the sum of values of negative value rules.

It is clear from Definition 4.13 that for a negative value rule r_x, each of its dummy rules r_x^i is always incompatible with r_x. This is because r_x^i and r_x always share the same positive literal a_1, but meanwhile r_x^i either takes a positive literal of r_x as its negative literal, or takes the negative literal of r_x as its positive literal. Thus the negative value rule is selected only if none of its dummy rules are selected, as described in Theorem 4.6. The rigorous proof is stated in [111].

Theorem 4.14. *The encoding given by Definition 4.7 and Definition 4.13 with transitive laws leads a MaxSAT solver to output the correct results of the CSG problem, consisting of a set of negative value rules.*

Proof. Hard clauses introduced by Definition 4.13 (1) and (2) are encoded from Theorem 4.6. Now, we prove the encoding is correct. Let R' be a feasible rule set and $\mathcal{B}'$ be the corresponding Boolean variable set. Let $D(r_x) = \{r_x^1, r_x^2, \ldots, r_x^{m-1}\}$ be the set of dummy rules created by a negative value rule r_x, and $B_{D(r_x)} = \{B_x^1, B_x^2, \ldots, B_x^{m-1}\}$ be the set of corresponding Boolean variables. Consider the hard clause introduced by Definition 4.13 (1). If $\forall r_x^i \in D(r_x)$, $r_x^i \notin R'$, i.e., $B_x^i \notin \mathcal{B}'$, we have to make $B_x \in \mathcal{B}'$ according to the hard clause introduced by Definition 4.13 (1), i.e., $r_x \in R'$. This agrees with the "if" part of Theorem 4.6. Next, we consider the hard clauses introduced by Definition 4.13 (2). If $r_x \in R'$, i.e., $B_x \in \mathcal{B}'$, then we have to make $B_x^i \notin \mathcal{B}'$ for $\forall B_x^i \in B_{D(r_x)}$ according to the hard clauses introduced by Definition 4.13 (2), i.e., $\forall r_x^i \in D(r_x)$, $r_x^i \notin R'$. This is consistent with the "only if" part of Theorem 4.6.

Under the above encoding, the CSG problem has been encoded into the one containing only positive value rules. The remaining work is the same as introduced in Definition 4.7. Detailed proof of this part is mentioned in Lemma 4.8, Lemma 4.9, and Theorem 4.10. □

It must be noted that the construction of dummy rules guarantees that each negative value rule is incompatible with all of its dummy rules, and such "incompatible" relation has been captured by Definition 4.7. Therefore, in practice, hard clauses in Definition 4.13 (2) is redundant and can be omitted safely.

After the encoding of negative value rules and the constraints on dummy rules, the remaining work of solving the problem is the same as handling positive value rules introduced in Section 4.3.1, i.e., identifying the rule relations between each pair of rules and generating corresponding hard clauses on rule relations, as well as hard clauses from transitive laws. Note that when identifying rule relations and representing the transitivity of rule relations, newly generated dummy rules should be taken into account. More specifically, let R_d be the set of dummy rules, then the various types of rule relations encoded in Definition 4.4 should be identified over $R \cup R_d$, and the number of hard clauses for representing the transitive laws should be extended to $(n + n_d)(n + n_d - 1)(n + n_d - 2)/2$, where n_d is the cardinality of R_d.

4.3.4 Encoding Negative Value Embedded Rules

The WPM encoding of negative value embedded rules is similar to that introduced in Section 4.3.3, both derived from Theorem 4.6.

Definition 4.15. (WPM encoding for negative value embedded rules). Let ER be a set of negative value embedded rules in the explicit form. For each negative value embedded rule $er \in ER$ ($er : I_0|I_1, \ldots, I_k \rightarrow w_{er}$, $w_{er} < 0$), where $I_i = a_1 \wedge a_2 \wedge \cdots \wedge a_l \wedge \neg a_{l+1} \wedge \cdots \wedge \neg a_m$, $i = 0, \ldots, k$. I_i generates the following two types of dummy rules:

- $r^x_{er,i}$: $a_1 \wedge \neg a_{x+1} \rightarrow 0$ where $1 \leq x \leq l-1$,
- $r^y_{er,i}$: $a_1 \wedge a_{y+1} \rightarrow 0$ where $l \leq y \leq m-1$,

where $r^x_{er,i}$ denotes the x-th dummy rules of a condition I_i in er.

For the negative value embedded rule er, the following clauses are introduced:

- one soft clause: $(\neg B_{er}, -w_{er})$,
- one hard clause: $B_{er} \vee \bigvee_{\substack{i \in \{0,\ldots,k\} \\ x \in \{1,\ldots,m\}}} B^x_{er,i}$,

where $B^x_{er,i}$ is a Boolean variable corresponding to $r^x_{er,i}$.

After encoding, the result output by a MaxSAT solver is $(-W_{neg})$ larger than the social welfare of the CSG problem in PFGs, where W_{neg} is the sum of values of negative value embedded rules.

Theorem 4.16. *The encoding given by Definition 4.11 and Definition 4.15 leads a MaxSAT solver to output the correct results of the CSG problem, consisting of a set of negative value embedded rules.*

Proof. The correctness of Definition 4.11 has been proved in Theorem 4.12. In the following, we prove the encoding in Definition 4.15 is correct.

Given a negative value embedded rule er, if er does not apply to any embedded coalition in CS, i.e., $B_{er} = 0$, the payoff generated by the soft clause $(\neg B_{er}, -w_{er})$ and the rule er is $(-w_{er})$ and 0, respectively. If er applies to some embedded coalition in CS, i.e., $B_{er} = 1$, the payoff generated by $(\neg B_{er}, -w_{er})$ and er is 0 and w_{er}, respectively. Clearly, in either case, the payoff of $(\neg B_{er}, -w_{er})$ is $(-w_{er})$ larger than that of er. Considering a coalition game of n negative value embedded rules, the social welfare after encoding is $(-W_{neg})$ larger than the original one, where W_{neg} is the sum of values of negative value embedded rules.

Next, we prove the hard clause introduced by Definition 4.15 is correctly encoded from Theorem 4.6. If $\forall i \in \{0, \ldots, k\}$ and $x \in \{1, \ldots, m\}$, $B^x_{er,i} = 0$, then, to satisfy the hard clause, B_{er} must be 1. This coincides with the "if" part of Theorem 4.6. If at least one $B^x_{er,i}$ is 1, then the value of B_{er} is unrestricted by the hard clause. In such a case, a MaxSAT solver prefers $B_{er} = 0$ because the payoff of $(-w_{er})$ can be gained. This is consistent with the converse-negative proposition of the "only if" part in Theorem 4.6. □

Example 4.8. *For $A = \{a_1, a_2, a_3, a_4\}$, assume there are four rules:*

$$r_1 : a_1 \wedge a_2 \rightarrow 2, \qquad r_2 : a_3 \wedge \neg a_4 \rightarrow 1,$$
$$r_3 : a_4 \wedge \neg a_1 \rightarrow -1, \qquad r_4 : a_1 \wedge a_2 \wedge \neg a_4 | a_4 \wedge \neg a_3 \rightarrow -1.$$

We introduce four weighted soft clauses $(B_1, 2)$, $(B_2, 1)$, $(\neg B_3, 1)$ and $(\neg B_4, 1)$, respectively. r_3 generates a dummy rule $r_5 : a_1 \wedge a_4 \rightarrow 0$. The constraint is specified by $B_3 \vee B_5$. r_4 generates 3 dummy rules: $r_6 : a_1 \wedge \neg a_2 \rightarrow 0$, $r_7 : a_1 \wedge a_4 \rightarrow 0$, $r_8 : a_4 \wedge a_3 \rightarrow 0$. The constraint is specified by $B_4 \vee B_6 \vee B_7 \vee B_8$. The remaining work is the same as handling positive value rules introduced in Section 4.3.1, i.e., identifying the rule relations between each pair of rules and generating corresponding hard clauses on rule relations, as well as hard clauses from transitive laws. To solve this problem, 36 variables and 143 clauses are generated. The generated clauses include 4 weighted soft clauses representing rules,

2 hard clauses generated by the negative value (embedded) rule, 32 hard clauses capturing the constraints on rules, and 105 hard clauses representing the transitive laws. The original social welfare is recovered by subtracting $(-W_{neg}) = 2$ *from the result output by the solver. Readers may refer to Appendix B for the complete file in WPM input format.*

4.4 Evaluation

This section shows the evaluation of the rule relation-based WPM approach (RWPM) for solving the CSG problem in CFGs (i.e., without externalities) and PFGs (with externalities), respectively. The contents of this section include generating problem instances, selecting appropriate MaxSAT solvers, and comparing the experimental results of our WPM encoding with the previous algorithm.

4.4.1 Generating Problem Instances

The method of generating the instances was described in [111], summarized as follows. First create a coalition with one random agent, then repeatedly add a new random agent with the probability of α until an agent is not added or the coalition includes all agents. Then, we repeatedly add a new condition of each rule with probability β until the condition is not added. The value of a rule is chosen between 0 and the number of agents in the rule, uniformly at random. In addition, for each coalition that contains more than one agent, we move an agent from positive to negative with the probability of p. Furthermore, we convert the value of a coalition from positive to negative with the probability of q.

The settings of parameters are given as follows.

- For MC-nets (i.e., the CSG problem in CFGs), we set $\alpha = 0.55$, $\beta = 0$, $p = 0.2$, $q = 0.2$, and $\#rules = \#agents$, ranging from 10 to 150. For each fixed $\#rules$, 100 problem instances are generated.
- For embedded MC-nets (i.e., the CSG problem in PFGs), we set $\alpha = 0.55$, $\beta = 0.15$, $p = 0.2$, $q = 0.2$, and $\#rules = \#agents$, ranging from 10 to 120. For each fixed $\#rules$, 100 problem instances are generated.

4.4.2 Selecting Appropriate Solvers

As mentioned before, RWPM encodes the CSG problem into propositional logic and needs a MaxSAT solver to output the final solution. Therefore, choosing the right MaxSAT solver in our experiment is of great importance. So far, there are a variety of MaxSAT solvers for a wide range of practical applications. They can be classified into two categories. The one implements a branch and bound scheme, and the other one uses a state-of-the-art SAT solver as an inference engine, referred to as SAT-based solver. SAT-based solvers are further classified into two: satisfiability-based and unsatisfiability-based. The state-of-the-art solvers for weighted partial MaxSAT, which we used, are listed as follows: Akmaxsat (version 1.1) [53] and WmaxSatz (version 2009) [59] are branch and bound solvers. Sat4j (version 2.2.3) [10] and ShinMaxSat][41] are satisfiability-based solvers. WPM1 (version 2012) [4] and Pwbo (version 2.0) [67] are unsatisfiability-based solvers. Note that Pwbo is a parallel solver incorporating a satisfiability-based search into an unsatisfiability-based approach. To select an appropriate solver in our evaluation, we apply these MaxSAT solvers to solve instances generated from MC-nets and embedded MC-nets, respectively shown in Tab. 4.1 and Tab. 4.2. For each instance and solver, there is a time limit of 900 seconds. Number in bracket means the number of instances that were successfully solved within the time limit by the corresponding solver and is omitted in the table if the solver managed to solve all the 100 instances. At $\#rules = N$, if a solver fails to solve any instances within the time limit, we terminate the solver and mark "/" for the corresponding solver with $\#rules > N$. The tests were carried out on a Core i5-2540 2.6GHz processor with 8GB RAM.

As can be seen from Tab. 4.1, Sat4j, ShinMaxSat, and Pwbo managed to solve all the problem instances with $\#rules$ ranging from 10 to 150. By contrast, branch and bound-based solvers performed worst in our experiment. In addition, as $\#rules$ increases, especially when $\#rules$ reaches 110, the superiority of Sat4j becomes more remarkable, and it can be inferred that Sat4j may be more desirable when $\#agents$ keeps growing. Similar analysis could be done to Tab. 4.2, which shows Sat4j finished solving all the instances with the shortest time. Therefore, we conclude that Sat4j is the appropriate solver for RWPM to solve the CSG problem in both CFGs and PFGs.

TABLE 4.1: Average wall-clock time (seconds) required for RWPM in MC-nets

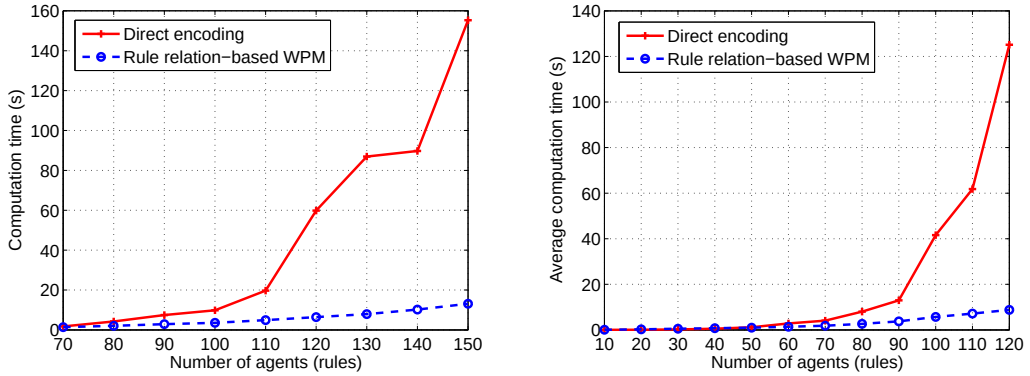
		Sat	Unsat	Sat/Unsat	Branch and Bound	
#rules	Sat4j	ShinMaxSat	WPM1	Pwbo	Akmaxsat	WmaxSatz
10	0.035	0.014	0.002	0.006	2.951	6.349(98)
20	0.191	0.490	0.032	0.028	6.380(73)	/
30	0.398	2.638	0.138	0.058	/	/
40	0.507	4.927	0.520	0.127	/	/
50	0.734	9.063	1.397(99)	0.301	/	/
60	1.058	14.321	/	0.477	/	/
70	1.391	19.269	/	0.826	/	/
80	2.098	28.630	/	1.584	/	/
90	3.012	37.074	/	2.725	/	/
100	3.849	43.797	/	3.521	/	/
110	5.197	58.272	/	5.412	/	/
120	6.707	72.284	/	7.816	/	/
130	8.729	87.563	/	10.036	/	/
140	11.189	103.199	/	15.085	/	/
150	13.283	125.880	/	17.352	/	/

TABLE 4.2: Average wall-clock time (seconds) required for RWPM in embedded MC-nets

		Sat	Unsat	Sat/Unsat	Branch and Bound	
#rules	Sat4j	ShinMaxSat	WPM1	Pwbo	Akmaxsat	WmaxSatz
10	0.055	0.021	0.005	0.010	5.172(97)	31.998(92)
20	0.258	0.738	0.066	0.031	/	/
30	0.461	2.715	0.306	0.095	/	/
40	0.627	5.999	0.822	0.218	/	/
50	0.947	10.950	2.265(95)	0.464	/	/
60	1.430	17.447	/	0.921	/	/
70	1.936	24.521	/	1.615	/	/
80	2.979	34.526	/	2.828	/	/
90	4.146	47.520	/	5.351	/	/
100	5.708	59.454	/	7.580	/	/
110	7.687	74.237	/	10.435	/	/
120	9.970	95.259	/	14.791	/	/

4.4.3 Results

In this subsection, we compare the RWPM encoding with the state-of-the-art optimization algorithm, i.e., direct encoding [75, 111], which does not make use of MaxSAT solvers.



(a) Computation time for MC-nets

(b) Computation time for embedded MC-nets

FIGURE 4.2: Average computation time of RWPM and direct encoding [75, 111]

Our test was run on a Core i5-2540 2.6GHz processor with 8GB RAM and employed Sat4j as the solver. By contrast, the direct encoding algorithm was run on an Xeon E5540 2.53GHz processor with 24GB RAM and used ILOG CPLEX (version 11.2) as a general-purpose mixed integer programming package.

Figure 4.2 depicts the average computation time for solving the generated problem instances in MC-nets and embedded MC-nets, by the direct encoding algorithm and RWPM. It is clear that RWPM outperforms the direct encoding algorithm for large values of $\#rules$. In particular, for MC-nets, as Fig. 4.2(a) shows, when $\#rules$ reaches over 110, the average computation time of direct encoding increases fast and surges over 150 seconds to solve instances with $\#rules = 150$. By contrast, the computation time of RWPM goes up much slowly with the increase of $\#agents$, and gets around 13 seconds at $\#agents = 150$. On the other hand, for embedded MC-nets, the average computation time of direct encoding soars sharply when $\#rules$ is larger than 90, and reaches around 125 seconds at $\#rules = 120$. In comparison, the computation time of RWPM goes up quite smoothly and is merely 10 seconds to solve the same set of instances at $\#rules = 120$, as depicted in Fig. 4.2(b).

4.5 Chapter Summary

In this chapter, we made the first step towards a study of applying MaxSAT solvers to the CSG problem for CFGs and PFGs, represented by MC-nets and embedded MC-nets, respectively. Specifically, with the encodings of rule relations, we showed that the CSG problem can scale up significantly when MaxSAT solvers are exploited, even though

the rule relation-based WPM approach is encoded directly from the previous algorithms. Experiments demonstrated the superiority of the WPM encoding. Specifically, instances for CFGs with 150 rules (agents) could be solved within 14 seconds on average, and those for PFGs with 120 rules (agents) could be solved within 10 seconds on average. In both cases, the computation time for solving the instances is largely reduced, compared to previous algorithms that do not make use of MaxSAT. This is our first attempt to apply an WPM encoding for solving the CSG problem.

Chapter 5

MaxSAT Encoding for the CSG Problem based on Agent Relations

Chapter 4 introduces a rule relation-based WPM encoding for solving the CSG problem and demonstrate the superiority of the WPM encoding to other advanced optimization techniques. This chapter seeks for a more efficient WPM encoding to solve the MC-net-based CSG problem. Instead of encoding rule relations into propositional logic, we capture a more fine-grained relation, i.e., the relations between each pair of agents, and reform the MC-net-based CSG problem as a set of extended WPM (EWPM) formulas. With the EWPM-to-WPM transformation proposed in Chapter 2, these EWPM formulas are easily converted to standard WPM formulas, which could be solved by existing WPM solvers.

This chapter is organized as follows. Section 5.1 describes the main idea of the agent relation-based WPM encoding. Specifically, Section 5.1.1 introduces the concept and the encoding of agent relations, which is the foundation of this chapter. Section 5.1.2 and 5.1.3 describes the agent relation-based WPM encoding of positive value (embedded) rules and negative value (embedded) rules, respectively. Evaluation is subject to Section 5.2. We compare the performance of the agent relation-based WPM encoding with that of the rule relation-based one, and try to explain the reason why one dominates the other one by investigating the number of variables and clauses generated during the problem solving. Finally, in section 5.3, we concludes this chapter.

5.1 WPM Encoding on Agent Relation

In a MC-net, a rule r is expressed in a syntactic form $I_r \to w_r$, where $w_r \in \mathbb{R}$ and I_r is the condition of rule r, denoted by a conjunction of literals over the set of agents A, i.e., $\{a_1 \wedge a_2 \wedge \cdots \wedge a_l \wedge \neg a_{l+1} \wedge \cdots \wedge \neg a_m\}$. $\{a_j\}_{j=1}^{l}$ are called *positive literals*, denoted by P_r, and $\{a_j\}_{j=l+1}^{m}$ are *negative literals*, denoted by N_r. A rule r is said to *apply to* a coalition C if $P_r \subseteq C$ and $N_r \cap C = \emptyset$, i.e., all agents in P_r are in C, and none of agents in N_r are in C.

Similar to Chapter 4, in this chapter, we assume each rule contains at least two agents, one of which is positive literal. This is because any rule of only one agent (suppose a) always applies to the coalition containing a, no matter how a coalition structure is structured.

5.1.1 Agent Relation

Let $\mathcal{A}_r$ be the set of agents in rule r, i.e., $\mathcal{A}_r = P_r \cup N_r$. Given a rule r of m agents, we call a_p *standard agent* if p is the minimum index of all positive literals in r, formally, $a_p \in P_r, \forall a_l \in P_r \setminus \{a_p\}, p < l$. With a_p, the agent relation between each pair of agents in r can be captured by the relations between a_p and a_k, $a_k \in \mathcal{A}_r \setminus \{a_p\}$. The agent relation between a_p and a_k is denoted by a Boolean variable $\mathcal{C}_{p,k}$ (if $p < k$) or $\mathcal{C}_{k,p}$ (if $p > k$). Intuitively, the value of the Boolean variable is equal to 1 if $a_k \in P_r$, and equal to 0 if $a_k \in N_r$.

For a rule r_i expressed as $\{a_1 \wedge a_2 \wedge \cdots \wedge a_l \wedge \neg a_{l+1} \wedge \cdots \wedge \neg a_m\}$, the standard agent is a_1. We interpret the condition of a rule by using the agent relations between the standard agent a_1 and other agents in the rule, showing in Definition 5.1.

Definition 5.1. (Agent relation-based MC-nets). An agent relation-based MC-net consists of a set of rules R. Each rule $r_i \in R$ is expressed in a syntactic form $\mathcal{S}_i \to w_i$, where $\mathcal{S}_i$ is a conjunction of agent relations, formally expressed by $\mathcal{C}_{1,2} \wedge \cdots \wedge \mathcal{C}_{1,l} \wedge \neg\mathcal{C}_{1,l+1} \wedge \cdots \wedge \neg\mathcal{C}_{1,m}$. A rule r_i is said to *apply* to a coalition C if $\mathcal{S}_i$ is true, i.e., $\{\mathcal{C}_{1,k}\}_{k=2}^{l}$ are all true, and $\{\mathcal{C}_{1,k}\}_{k=l+1}^{m}$ are all false.

The definition of the agent relation-based MC-nets could be extended to the embedded MC-nets, as Definition 5.2 shows. Each embedded rule is expressed in explicit form defined in Section 4.3.2.

Definition 5.2. (Agent relation-based embedded MC-nets). An agent relation-based embedded MC-net consists of a set of embedded rules ER. Each rule $er \in ER$ is expressed in a syntactic form $\mathcal{S}_0 \wedge \mathcal{S}_1 \wedge \ldots \wedge \mathcal{S}_k \rightarrow w_{er}$, where each $\mathcal{S}_i$: $i \in \{0, \ldots, k\}$ is a conjunction of agent relations, as defined in Definition 5.1. An embedded rule er is said to apply to an embedded coalition (C, CS) if $\{\mathcal{S}_i\}_{i=0}^{k}$ are all true.

Apparently, the relation that two agents are in the same coalition is transitive, i.e., any three agents a_i, a_j, and a_k $(1 \leq i < j < k \leq m)$ satisfy the following transitive laws:

- $\neg\mathcal{C}_{i,j} \vee \neg\mathcal{C}_{j,k} \vee \mathcal{C}_{i,k}$,
- $\neg\mathcal{C}_{i,j} \vee \neg\mathcal{C}_{i,k} \vee \mathcal{C}_{j,k}$,
- $\neg\mathcal{C}_{i,k} \vee \neg\mathcal{C}_{j,k} \vee \mathcal{C}_{i,j}$,

where m is the number of agents. The number of hard clauses for representing the transitive laws is $m \cdot (m-1) \cdot (m-2)/2$.

Example 5.1. *For $A = \{a_1, a_2, a_3, a_4\}$, assume there are four rules:*

$$r_1 : a_1 \wedge a_2 \rightarrow 2, \qquad r_2 : a_3 \wedge \neg a_4 \rightarrow 1,$$
$$r_3 : a_4 \wedge \neg a_1 \rightarrow 1, \qquad r_4 : a_1 \wedge a_2 \wedge \neg a_4 | a_4 \wedge \neg a_3 \rightarrow 1.$$

In agent relation-based MC-net, these four rules are expressed as follows:

$$r_1 : \mathcal{C}_{1,2} \rightarrow 2, \qquad r_2 : \neg\mathcal{C}_{3,4} \rightarrow 1,$$
$$r_3 : \neg\mathcal{C}_{1,4} \rightarrow 1, \qquad r_4 : \mathcal{C}_{1,2} \wedge \neg\mathcal{C}_{1,4} \wedge \neg\mathcal{C}_{3,4} \rightarrow 1.$$

The four agents totally generate 12 hard clauses by the transitive laws.

The transitive laws introduced above could be refined in some cases. Take the example of a CSG problem with three rules: $r_1 : a_1 \wedge a_2 \rightarrow 2$, $r_2 : a_3 \wedge \neg a_4 \wedge \neg a_5 \rightarrow 1$, $r_3 : a_1 \wedge \neg a_6 \rightarrow 1$. Agents a_3, a_4 and a_5 only appear in r_2 and have nothing to do with a_1, a_2 and a_6. Then we can combine a_3, a_4 and a_5 into a group, and make the rest three agents form the other group. Obviously, these two groups are independent of each other. Thus, the number of clauses for representing transitive laws is calculated within two groups respectively. In this example, the number of transitive laws is only 6, while the original one sums up to $6 \cdot 5 \cdot 4/2 = 60$.

5.1.2 Encoding Positive Value (Embedded) Rules

With EWPM-to-WPM transformation and the agent relations, rules in MC-nets and embedded MC-nets could be encoded in straightforward way. Specifically, encodings of positive value (embedded) rules are given in this subsection and those for negative value (embedded) rules are introduced in the next subsection.

Definition 5.3. (WPM encoding for positive value rules). For a positive value rule $r_i : \mathcal{S}_i \to w_i$ $(w_i > 0)$, where $\mathcal{S}_i = \mathcal{C}_{1,2} \wedge \cdots \wedge \mathcal{C}_{1,l} \wedge \neg\mathcal{C}_{1,l+1} \wedge \cdots \wedge \neg\mathcal{C}_{1,m}$, the following clauses are introduced, where u_i is a new Boolean variable.

(i) a weighted soft clause: (u_i, w_i), and
(ii) $(m-1)$ hard clauses: $\neg u_i \vee \mathcal{C}_{1,2}, \ldots, \neg u_i \vee \mathcal{C}_{1,l}, \neg u_i \vee \neg\mathcal{C}_{1,l+1}, \ldots, \neg u_i \vee \neg\mathcal{C}_{1,m}$.

Definition 5.3 is in accordance with EWPM-to-WPM transformation, which leads the following theorem to hold.

Theorem 5.4. *The encoding given by Definition 5.3 with the transitive laws leads a MaxSAT solver to output the correct results of the CSG problem, which consists of a set of positive value rules.*

Proof. If there is a coalition structure CS, then we can make an assignment $\mathcal{A}$ which satisfies all hard clauses for the transitive laws so as to agree with CS. Reversely, if there is an assignment $\mathcal{A}$ which satisfies all hard clauses for the transitive laws, then we obtain a coalition structure CS which agrees with $\mathcal{A}$. Thus, there is a one-to-one correspondence between a set of coalition structures and a set of assignments which satisfy all hard clauses for the transitive laws.

Definition 5.3. (i) and (ii) correspond to Definition 3.1. (i) and (iii), respectively. According to Theorem 3.4 and Proposition 1, Definition 3.1. (i) and (iii) guarantee the correctness of EWPM-to-WPM transformation for positive soft formulas. Therefore, MaxSAT solvers calculate the correct social welfare. □

The encoding for positive value embedded rules can be fulfilled in the similar way.

Definition 5.5. (WPM encoding for positive value embedded rules). For a positive value embedded rule $er : \mathcal{S}_0 \wedge \mathcal{S}_1 \wedge \cdots \wedge \mathcal{S}_k \to w_{er}$ $(w_{er} > 0)$, the following clauses are introduced, where u_{er} is a new Boolean variable.

(i) a weighted soft clause: (u_{er}, w_{er}), and

(ii) $k+1$ propositional Boolean formulas that could be expanded to several hard clauses: $\neg u_{er} \vee \mathcal{S}_0, \ldots, \neg u_{er} \vee \mathcal{S}_1, \neg u_{er} \vee \mathcal{S}_k$.

Theorem 5.6. *The encoding given by Definition 5.5 with the transitive laws leads a MaxSAT solver to output the correct results of the CSG problem, which consists of a set of positive value embedded rules.*

Proof. Let CS be a coalition structure. We can make an assignment $\mathcal{A}$ which satisfies all hard clauses for the transitive laws so as to agree with CS.

If a positive value embedded rule er applies to an embedded coalition in CS, then $\mathcal{A}$ satisfies each $\mathcal{S}_i$, and the corresponding soft clause (u_{er}, w_{er}) is not restricted by any hard clauses. In such a case, MaxSAT solvers prefer $u_{er} = 1$ to $u_{er} = 0$ because the payoff w_{er} can be gained.

If r_{er} does not apply to any embedded coalition in CS, then $\mathcal{A}$ does not satisfy at least one $\mathcal{S}_i$ unless $u_{er} = 0$. Thus, the corresponding soft clause (u_{er}, w_{er}) is unsatisfied, that is, the payoff is 0.

In either case, MaxSAT solvers calculate the correct social welfare. □

Example 5.2. *In Example 5.1, the four rules are encoded into the following soft and hard clauses.*

- $r_1 : \mathcal{C}_{1,2} \to 2$ *is encoded into a soft clause* $(u_1, 2)$ *and a hard clause* $\neg u_1 \wedge \mathcal{C}_{1,2}$,
- $r_2 : \neg\mathcal{C}_{3,4} \to 1$ *is encoded into a soft clause* $(u_2, 1)$ *and a hard clause* $\neg u_2 \vee \neg\mathcal{C}_{3,4}$,
- $r_3 : a_4 \wedge \neg a_1 \to 1$ *is encoded into a soft clause* $(u_3, 1)$ *and a hard clause* $\neg u_3 \vee \neg\mathcal{C}_{1,4}$,
- $r_4 : a_1 \wedge a_2 \wedge \neg a_4 | a_4 \wedge \neg a_3 \to 1$ *is encoded into a soft clause* $(u_4, 1)$ *and three hard clauses:* $\neg u_4 \vee \mathcal{C}_{1,2}$, $\neg u_4 \vee \neg\mathcal{C}_{1,4}$, $\neg u_4 \vee \neg\mathcal{C}_{3,4}$,

where $u_i\,(i = 1, \ldots, 4)$ *are newly introduced Boolean variables.*

To solve this problem instance, a total of 10 variables and 22 clauses are generated. These clauses include: 4 weighted soft clauses representing rules, 6 hard clauses encoded from conjunction formulas, and 12 hard clauses representing transitive laws. For the complete file in WPM input format, readers may refer to Appendix C.

5.1.3 Encoding Negative Value (Embedded) Rules

If a rule is a negative value rule $r_i : \mathcal{S}_i \to w_i$ $(w_i < 0)$, as explained in previous section, the EWPM-to-WPM transformation referred to Definition 3.1. (i) and (ii), which derive the following encodings.

Definition 5.7. (WPM encoding for negative value rules). For each negative value rule $r_i : \mathcal{S}_i \to w_i$ $(w_i < 0)$, the following clauses are introduced, where u_i is a new Boolean variable.

(i) a weighted soft clause: $(\neg u_i, -w_i)$, and
(ii) a hard clause: $\neg \mathcal{S}_i \vee u_i$.

The social welfare after encoding is $(-W_{neg})$ larger than the original one, where W_{neg} is the sum of negative weights.

Theorem 5.8. *The encoding given by Definition 5.7 with the transitive laws leads a MaxSAT solver to output the correct results of the CSG problem, which consists of a set of negative value rules.*

Proof. Let CS be a coalition structure. We can make an assignment $\mathcal{A}$ which satisfies all hard clauses for the transitive laws so as to agree with CS.

Definition 5.7. (i) and (ii) correspond to Definition 3.1. (i) and (ii), respectively. According to Theorem 3.4 and Proposition 1, Definition 3.1. (i) and (ii) guarantee the correctness of EWPM-to-WPM transformation for negative soft formulas. Therefore, MaxSAT solvers calculate the correct social welfare. □

Definition 5.9. (WPM encoding for negative value embedded rules). For each negative value embedded rule $er : \mathcal{S}_0 \wedge \mathcal{S}_1 \wedge \cdots \wedge \mathcal{S}_k \to w_{er}$ $(w_{er} < 0)$, the following clauses are introduced, where u_i is a new Boolean variable.

(i) a weighted soft clause: $(\neg u_{er}, -w_{er})$, and
(ii) a hard clause: $\neg \mathcal{S}_0 \vee \neg \mathcal{S}_1 \vee \cdots \vee \neg \mathcal{S}_k \vee u_i$.

The social welfare after encoding is $(-W_{neg})$ larger than the original one, where W_{neg} is the sum of negative weights.

Theorem 5.10. *The encoding given by Definition 5.9 with the transitive laws leads a MaxSAT solver to output the correct results of the CSG problem, which consists of a set of negative value embedded rules.*

Proof. Let CS be a coalition structure. We can make an assignment $\mathcal{A}$ which satisfies all hard clauses for the transitive laws so as to agree with CS.

If a negative value embedded rule r_{er} applies to an embedded coalition in CS, then $\mathcal{A}$ satisfies each $\mathcal{S}_i$. This implies $\mathcal{A}$ does not satisfy the corresponding hard clause unless $u_{er} = 1$. Thus, the corresponding soft clause $(\neg u_{er}, -w_{er})$ is unsatisfied. That is, no payoff is obtained with MaxSAT solvers. In contrast, the payoff of r_{er} is w_{er} because r_{er} applies to an embedded coalition in CS.

If a negative value rule r_{er} does not apply to any embedded coalition in CS, then $\mathcal{A}$ does not satisfy at least one $\mathcal{S}_i$. This implies $\mathcal{A}$ satisfies the corresponding hard clause, and the corresponding soft clause $(\neg u_{er}, -w_{er})$ is not restricted by any hard clauses. In such a case, MaxSAT solvers prefer $u_{er} = 0$ because the payoff of $-w_{er}$, which is positive, is gained. In contrast, the payoff of r_{er} is 0 because r_{er} does not apply to any coalition in CS.

In either case, MaxSAT solvers overestimate the social welfare by $-w_{er}$ for a negative value embedded rule, thus the value after encoding is $(-W_{neg})$ larger than the original one. □

Example 5.3. *For $A = \{a_1, a_2, a_3, a_4\}$, assume there are four rules:*

$$r_1 : a_1 \wedge a_2 \to 2, \qquad r_2 : a_3 \wedge \neg a_4 \to 1,$$
$$r_3 : a_4 \wedge \neg a_1 \to -1, \qquad r_4 : a_1 \wedge a_2 \wedge \neg a_4 | a_4 \wedge \neg a_3 \to -1.$$

r_1 and r_2 are encoded the same way as shows in Example 5.2. r_3 and r_4 are encoded as follows.

- *$r_3 : a_4 \wedge \neg a_1 \to -1$ is encoded into a soft clause $(\neg u_3, 1)$ and a hard clause $\mathcal{C}_{1,4} \vee u_3$,*
- *$r_4 : a_1 \wedge a_2 \wedge \neg a_4 | a_4 \wedge \neg a_3 \to -1$ is encoded into a soft clause $(\neg u_4, 1)$ and a hard clause $\neg \mathcal{C}_{1,2} \vee \mathcal{C}_{1,4} \vee \mathcal{C}_{3,4} \vee u_4$,*

where u_3 and u_4 are newly introduced Boolean variables.

To solve this problem, a total of 10 variables and 20 clauses are generated. These clauses include: 4 weighted soft clauses representing rules, 4 hard clauses encoded from conjunction formulas, and 12 hard clauses representing transitive laws. Readers may refer to Appendix D for the complete file in WPM format.

5.2 Evaluation

In this section, we experimentally evaluated the performance of the rule relation-based WPM approach (RWPM) and agent relation-based WPM approach (AWPM). RWPM is derived directly from the existing algorithms [75, 111], as introduced in Chapter 4, and AWPM is a brand-new encoding described in this chapter.

The method of generating problem instances is the same as that introduced in Section 4.4.1. In the following, we investigate which MaxSAT solver is suitable for AWPM in MC-nets and embedded MC-nets, respectively. The solvers evaluated in our experiments include: branch and bound solver Akmaxsat (version 1.1) [53] and WmaxSatz (version 2009) [59], satisfiability-based solver Sat4j (version 2.2.3) [10] and ShinMaxSat [41], unsatisfiability-based solver WPM1 (version 2012) [4] and Pwbo (version 2.0) [67]. Table 5.1 and Tab. 5.2 show the comparison results of their performances in MC-nets and embedded MC-nets, respectively. Tests in Tab. 5.1 were carried out on a Core i7-2600 3.40GHz processor with 8GB RAM, and tests in Tab. 5.2 were carried out on a Core i5-2540 2.6GHz processor with 8GB RAM. Consistent with the settings for RWPM, for each instance and solver, there is a time limit of 900 seconds. Number in bracket means the number of instances that were successfully solved within the time limit by the corresponding solver and is omitted in the table if the solver managed to solve all the 100 instances. At $\#agents = N$, if a solver fails to solve all instances within the time limit, then we terminate the solver and mark "/" for the corresponding solver with $\#agents > N$.

As can be seen from Tab. 5.1, Sat4j, ShinMaxSat, and Pwbo managed to solve all the problem instances. Among these three solvers, Pwbo outperformed others when $\#agents$ is no greater than 120, while Sat4j becomes more desiable when $\#agents$ increases over 120. We can predict that, the superiority of Sat4j would be more remarkable when $\#agents$ keeps growing. Thus we choose Sat4j for the evaluation of AWPM in MC-nets. Similar analysis could be done on Tab. 5.2, where Pwbo performed best with $\#agents$ ranging from 10 to 120.

TABLE 5.1: Average wall-clock time (seconds) required for AWPM in MC-nets

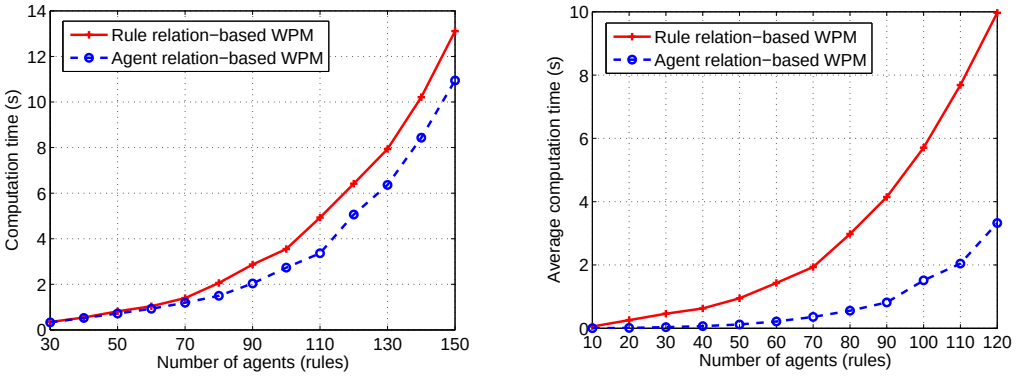
	Sat		Unsat	Sat/Unsat	Branch and Bound	
#agents	Sat4j	ShinMaxSat	WPM1	Pwbo	Akmaxsat	WmaxSatz
10	0.13	0.02	0.01	0.01	0.02	0.03
20	0.20	0.28	0.03	0.02	0.44	0.25
30	0.28	1.27	0.07(99)	0.03	13.70	9.18
40	0.33	2.75	/	0.05	45.64(99)	46.03(89)
50	0.40	4.61	/	0.08	/	/
60	0.50	6.67	/	0.14	/	/
70	0.62	9.14	/	0.21	/	/
80	0.77	12.06	/	0.30	/	/
90	1.02	16.22	/	0.53	/	/
100	1.36	20.40	/	0.75	/	/
110	1.67	23.31	/	1.25	/	/
120	2.47	30.57	/	2.04	/	/
130	3.21	34.50	/	2.91	/	/
140	4.38	44.95	/	5.26	/	/
150	5.79	51.00	/	6.61	/	/

TABLE 5.2: Average wall-clock time (seconds) required for AWPM in embedded MC-nets

	Sat/Unsat	Sat		Unsat	Branch and Bound	
#agents	Pwbo	Sat4j	ShinMaxSat	WPM1	Akmaxsat	WmaxSatz
10	0.001	0.025	0.008	0.001	0.006	0.001
20	0.010	0.101	0.569	0.020	0.716	0.389
30	0.033	0.298	1.912	0.070	13.669	8.192
40	0.065	0.447	4.923	0.188 (99)	123.023 (98)	89.412 (89)
50	0.120	0.489	7.833	/	/	/
60	0.213	0.737	11.859	/	/	/
70	0.359	0.945	17.952	/	/	/
80	0.556	1.192	21.889	/	/	/
90	0.815	1.476	29.752	/	/	/
100	1.513	2.338	38.129	/	/	/
110	2.043	2.775	46.575	/	/	/
120	3.324	3.887	55.976	/	/	/

Therefore, in what follows, we employ Sat4j and Pwbo as the solver to evaluate AWPM in MC-nets and embedded MC-nets, respectively. In comparison, the solver chosen for RWPM is Sat4j, which performed best according to our previous experiment in Chapter 4.4.2. All the tests below were run on a Core i5-2540 2.6GHz processor with 8GB RAM.

Figure 5.1 depicts the average computation time of RWPM and AWPM, for solving



(a) Computation time for MC-nets

(b) Computation time for embedded MC-nets

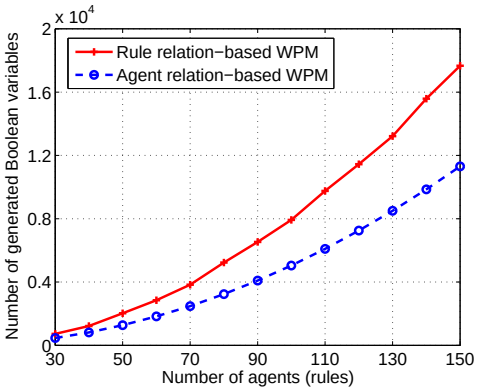
FIGURE 5.1: Average computation time of RWPM and AWPM

the CSG problem in MC-nets and embedded MC-nets, respectively. It is clear that AWPM is more time efficient than RWPM, and the superiority of AWPM becomes more remarkable as $\#agents$ goes up. Especially, in MC-nets, as Fig 5.1(a) shows, when $\#agnets$ increases to 150, the computation time for solving problem instances by AWPM is less than 11 seconds, while gets around 14 seconds for RWPM. In embedded MC-nets, as Fig 5.1(b) depicts, when $\#agents$ reaches 120, the computation time for solving problem instances by AWPM approach is less than 4 seconds, while reaches around 9 seconds for RWPM.

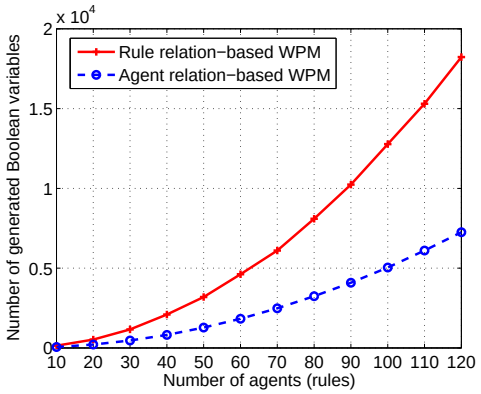
We explain the reason why AWPM is more time efficient by investigating the number of Boolean variables and clauses in these two approaches, depicted in Fig. 5.2 and Fig. 5.3, respectively. Obviously, both the numbers of variables and clauses in RWPM are greater than that in AWPM. In general, the computation time goes up as the number of variables and clauses increases. That is why RWPM needs more computation time than AWPM in our experiment.

5.3 Chapter Summary

In this chapter, in order to further improve the performance of solving the CSG problem, we exploited agent relations to encode the CSG problem into a set of Boolean formulas, which could be formulated as WPM instances with the EWPM-to-WPM transformation introduced in Chapter 3. Experiments showed that the agent relation-based WPM (AWPM) approach achieved higher efficiency than the rule relation-based WPM (RWPM). To be more specific, in MC-nets, problem instances with 150 agents (rules) were

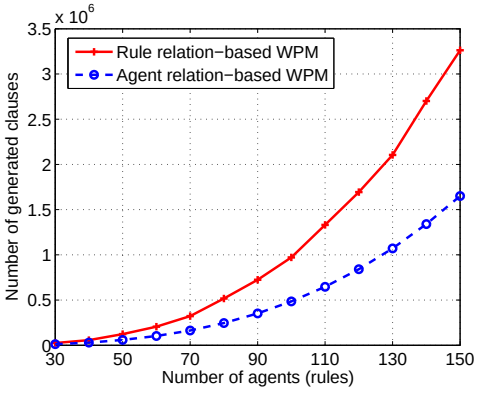


(a) No. of Boolean variables for MC-nets

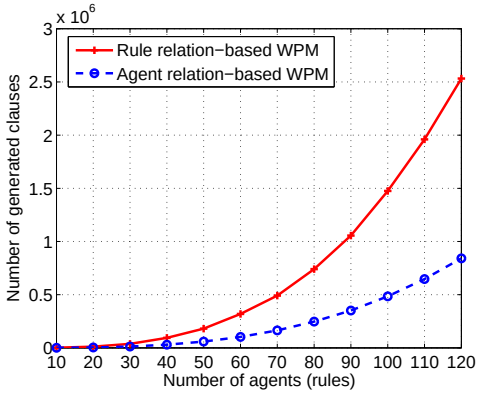


(b) No. of Boolean variables for embedded MC-nets

FIGURE 5.2: Number of Boolean variables in RWPM and AWPM



(a) No. of clauses for MC-nets



(b) No. of clauses for embedded MC-nets

FIGURE 5.3: Number of clauses in RWPM and AWPM

solved within 11 seconds by AWPM, while RWPM required 14 seconds on average. In MC-nets, instances with 120 agents (rules) were solved by AWPM within averagely 4 seconds. This is only a half of the time consumed by RWPM. Moreover, we examined the number of generated Boolean variables and clauses in these two methods. Results showed that, in both MC-nets and embedded MC-nets, RWPM generates more variables and clauses than AWPM to solve the same problem. Generally, the computation time of a solver goes up as the number of variables and clauses increases. That is why RWPM needs more time than AWPM to solve the same set of problem instances.

Chapter 6

MaxSAT Encoding for Recovering AES Key Schedules

Cold boot attack is a side channel attack that recovers data from memory, which persists for a short period after power is lost. In the course of this attack, the memory gradually degrades over time and only a corrupted version of the data may be available to the attacker. Recently, great efforts have been devoted to reconstructing the original data from a corrupted version of AES key schedules, based on the assumption that all bits in the charged states tend to decay to the ground states while no bit in the ground state ever inverts. Considering the relations among AES key bits are naturally expressed with a set of Boolean formulas, the previous work [49] has formulated these Boolean formulas, as well as bits in the charged states, as sets of hard clauses for a SAT solver. Then the SAT solver could infer other unknown (i.e., corrupted) bits and recover the AES key schedule reliably, as long as all the bits in the charged states are guaranteed to be correct. However, in practice, there is a small number of bits flipping in the opposite direction, i.e., the bits flip from the ground state flipping to the charged state. We call them *reverse flipping errors*. In this chapter, motivated by the previous work that formulates the relations of AES key bits as a SAT problem, we move one step further by taking the reverse flipping errors into consideration and employ an off-the-shelf MaxSAT solver to accomplish the key recovery of AES-128 key schedules from decayed memory images. This work can adapt well to real situations where decaying and reverse flipping errors co-exist.

This chapter is organized as follows. Section 6.1 briefly introduces the cold boot attack and AES key expansion. Section 6.2 provides an overview of the related works on AES

key recovery. Section 6.3 models the AES key bits from the key expansion algorithm. In Section 6.4, we describe the SAT encoding and MaxSAT encoding for recovering the AES key schedules from a true cold boot attack. Experiment and comparison are illustrated in Section 6.5 and chapter summary is given in Section 6.6.

6.1 Cold Boot Attack and AES

A dynamic random access memory (DRAM) cell is essentially a capacitor that can be either charged or discharged, indicating that a bit is in *the charged state* or *the ground state*, respectively. DRAM remanence, presented by Halderman in [34], refers to that after power is lost, the DRAM holds its state for several seconds, and for minutes or even hours if the chips are kept at low temperatures. Cold boot attack [34] is a sophisticated side channel attack that exploits DRAM remanence effects to recover sensitive data from a running computer. It poses a particular threat to systems that typically store sensitive data in memory. For example, several disk encryption systems have been defeated by the cold boot attack, including BitLocker, TrueCrypt, FileVault, LoopAES, and dm-crypt [37]. These on-the-fly disk encryption softwares typically store the encryption key in DRAM while the disk is mounted, which opens a door for an attacker with access with the contents of DRAM to learn the key and decrypt the disk.

The cold boot attack comes in three variants of increasing resistance to countermeasures [37]. The simplest is to reboot the machine and launch a custom kernel with a small memory footprint that gives the adversary access to the residual memory. A more advanced attack is to briefly cut power to the machine, then restore power and boot a custom kernel. This deprives the operating system of any opportunity to erase memory before shutting down. An even stronger attack is to cut the power, transplant the DRAM modules to a second PC prepared by the attacker, and use it to extract their state. This attack further deprives the original BIOS and PC hardware of any chances to clear the memory on boot.

Given the nature of the cold boot attack, memory bits gradually decay over time once power is removed, and finally, only a corrupted image of memory contents may be available to the attacker. The recovery of a cryptographic key from a corrupted image of memory contents is usually achieved by exploiting the redundancy of key material inherent in cryptographic algorithms. In practice, many encryption programs store data pre-computed from the encryption keys to speed up computation. For block ciphers, a

key schedule, made up of multiple roundkeys, is usually pre-computed from the secret key. This data contains much more information than the key itself, by which one can efficiently reconstruct the original key even in the presence of errors [37]. The focus of this chapter is to recover AES encryption keys from a cold boot attack.

Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001, and is currently used as a worldwide prevalent symmetric cryptographic algorithm. As a kind of block ciphers, AES is vulnerable to the cold boot attack, via which the attacker could extract the encryption key from memory. An AES encryption key refers to a key schedule consisting of multiple roundkeys that are expanded from an *initial key*, via the key expansion algorithm [31]. The length of an AES initial key is 128, 192, or 256 bits, referred to as AES-128, AES-192, AES-256, respectively. The AES key schedule is the primary source of key redundancy, which enables an attacker to reconstruct the initial key by exploiting the known bits present in the memory, even if the content he extracts has a moderate amount of errors.

6.2 Reltaed Works

This section first introduces the models related to decay patterns and then surveys existing works for recovering the AES key schedule from a cold boot attack.

When memory is out of power, the refresh cycle of DRAM is interrupted. Halderman et al. [34] observed that most memory bits tend to decay to the ground states as time goes on, with a constant and small fraction of bits flipping to the charged states. The ground state could be encoded as either 0 or 1, depending on how the cell is wired, thus the decay of memory bits is overwhelmingly either $0 \rightarrow 1$ or $1 \rightarrow 0$. The decay direction in a given region could be inferred by comparing the numbers of 1s and 0s since in an uncorrupted key schedule, we expect these to be approximately equal [34]. For simplicity of elaboration, in the rest of this chapter, we assume 1 as the charged state, and 0 as the ground state. Then the decay direction is overwhelmingly $1 \rightarrow 0$, with a small fraction from 0 to 1.

We denote the probability of 1 degrading to 0 by decay factor δ_0, and the probability of 0 flipping to 1 by some fixed δ_1. Generally speaking, δ_0 reflects the extent of decay, which approaches to 1 as time goes on after power is lost. By contrast, δ_1 is relatively

constant and tiny, from 0.05% to 0.1% [34]. According to the different settings of δ_1, existing works model the decay patterns by either of the following two cases.

- Perfect assumption: since δ_1 is quite small in true cold boot attacks, the decay direction is assumed only from 1 to 0 with no bit flipping in the opposite direction, i.e., δ_1 is set 0. Therefore, all 1s present in the decayed key schedule are correct with absolute certainty.
- Realistic assumption: δ_1 is from 0.05% to 0.1%. In this setting, both $1 \rightarrow 0$ and $0 \rightarrow 1$ coexist in an attack, thus none of the key bits in the decayed key schedule are known with absolute certainty.

The technique of identifying AES keys in memory has been developed in [37]. In the following, we investigate previous works on how to reconstruct AES key schedules after they have been extracted from memory.

The AES key schedule contains a large amount of linearity, which allows one to search for a small set of keys exhaustively and then combine these small pieces into the overall key. The method proposed in [34] takes advantage of the high amount of linearity. Instead of trying to reconstruct the entire AES-128 key at a time, their algorithm cuts up the 128-bit roundkey into four subsets of 32 bits, and uses 24 bits of the subsequent roundkey as redundancy. These small sets are decoded in order of likelihood, and then combined into a candidate key, which could be checked against the full schedule. Their work reconstructed AES-128 keys with $\delta_0 = 15\%$ and $\delta_1 = 0.1\%$ in a fraction of a second, and up to half of keys with $\delta = 30\%$ and $\delta_1 = 0.1\%$ within 30 seconds.

As far as we concern, [34] is the only work that works for the realistic assumption, while many existing works adopt the perfect assumption.

The algorithm presented in [109] makes better use of the AES key schedule structure, by modeling the search of keys in a depth-first tree under tree-pruning constraints. Their method allows one to recover the AES-128 key schedules with on average 300 seconds for $\delta_0 = 70\%$ and $\delta_1 = 0$. Although it was noted in [109] that the proposed algorithm did still work for realistic assumption, the methodology was not mentioned in their work, and the performance in this case was not demonstrated in [109]. Later on, motivated by the dramatic speed-up of Boolean Satisfiability (SAT) solvers, Abdel et al. [49] took the initial step to model the AES key recovery problem as a SAT problem by making full use of the present bits equal to 1. Based on the perfect assumption, all 1s in the decayed

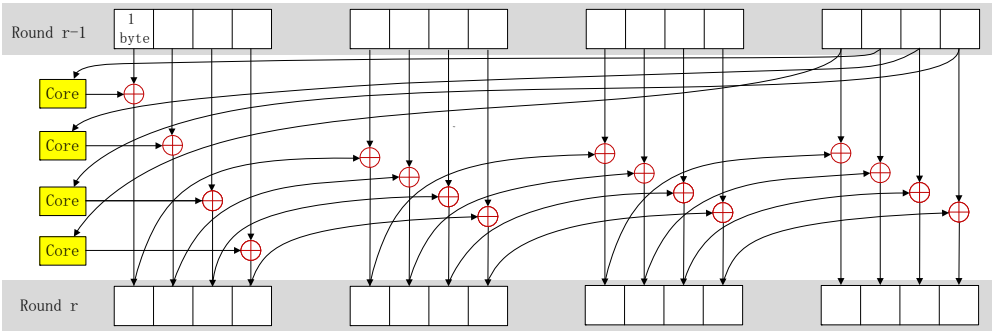


FIGURE 6.1: Diagram of AES-128 key expansion in adjacent two rounds

key schedule are correct, thus a set of correct constraints could be constructed from 1s. On the other hand, the large amount of redundant information available in the AES key schedule could also be formulated as constraints that have to be satisfied in a SAT problem. As a result, by employing CryptoMiniSat [104], an XOR supported SAT solver, their approach considerably improved the performance of AES key recovery, in terms of both the recovery speed and the maximum recoverable decay factor. Specifically, the authors reported that recovering AES-128 key schedules could be fulfilled with $\delta_0 = 70\%$ and $\delta_1 = 0$ in around 1.2 seconds on average, and recovering keys with $\delta_0 = 80\%$ and $\delta_1 = 0$ became possible.

6.3 Modeling Bits in AES-128 Key Schedules

AES is a widely spread symmetric key algorithm that uses the same cryptographic keys for both encryption and decryption. In this chapter, we focus on AES-128 and our method could be extended to AES-192 and AES-256 in a straightforward way since the key schedule for 128-bit, 192-bit, and 256-bit encryption are very similar, with only some constants changed.

An AES-128 key schedule consists of 11 roundkeys, each made up of 128 bits. The 0^{th} roundkey is equal to the initial key itself, which is bijectively mapped to the subsequent 10 roundkeys, via the public AES key expansion algorithm. Each bit in the key schedule, either 0 or 1, is naturally expressed as a Boolean variable. We denote the i^{th} bit of the r^{th} roundkey by b_i^r, where $0 \leqslant r \leqslant 10$ and $0 \leqslant i \leqslant 127$. $b_i^r = 1$ if the corresponding bit in the memory is 1 and $b_i^r = 0$ otherwise. The key schedule components are addressed with the following two notations: the round-dependent word array and the substitution box. The round-dependent word array, denoted by $R(r)$, contains the values $\{02\}_H^{r-1}$ for the

least significant byte and 0 for the rest bytes, with $\{02\}_H^{r-1}$ being powers of $\{02\}_H$ in the Galois field $GF\left(2^8\right)$, where $\{02\}_H$ is the hexadecimal representation of 2. Particularly, $R\left(r\right)$ $(1 \leqslant r \leqslant 10)$ is listed as follows.

$$
\begin{aligned}
R\left(1\right) &= \underbrace{0\cdots0}_{24\text{ bits}}00000001, & R\left(6\right) &= \underbrace{0\cdots0}_{24\text{ bits}}00100000,\\
R\left(2\right) &= \underbrace{0\cdots0}_{24\text{ bits}}00000010, & R\left(7\right) &= \underbrace{0\cdots0}_{24\text{ bits}}01000000,\\
R\left(3\right) &= \underbrace{0\cdots0}_{24\text{ bits}}00000100, & R\left(8\right) &= \underbrace{0\cdots0}_{24\text{ bits}}10000000,\\
R\left(4\right) &= \underbrace{0\cdots0}_{24\text{ bits}}00001000, & R\left(9\right) &= \underbrace{0\cdots0}_{24\text{ bits}}00011011,\\
R\left(5\right) &= \underbrace{0\cdots0}_{24\text{ bits}}00010000, & R\left(10\right) &= \underbrace{0\cdots0}_{24\text{ bits}}00110110.
\end{aligned}
$$

The substitution box, abbreviated with S-box, is a basic component regarding security in AES key schedule, which operates independently on a byte, by taking the multiplicative inverse in the finite field $GF\left(2^8\right)$ using the irreducible polynomial $x^8+x^4+x^3+x+1$ and then applying an affine transformation over $GF\left(2\right)$. For more details, we refer readers to the literature [31]. According to the hardware implementation of the AES key expansion algorithm, an S-box operation could be split into eight functions in algebraic normal form (ANF)[1], with 1-byte input and 1-bit output [117].(See Appendix E for a sample of Sbox expressed in ANF.) We denote each of the eight functions by $S_x\left(B_i^r\right)$, where x $(=0,\ldots,7)$ is the index of the function, and B_i^r is an input byte starting with b_i^r, following the least-significant-bit-first convention, i.e., $B_i^r=\{b_i^r,b_{i+1}^r,\ldots,b_{i+7}^r\}$. The output of an S-box is then obtained by combining the outputs of these eight functions into a byte, with the decreasing significance of x. Specifically, each bit in 1-10 roundkeys is described by the following formulas.

$$
\begin{aligned}
b_i^r &= b_i^{r-1} \oplus S_{i\,mod\,8}\left(B_{104+8\cdot\lfloor i/8\rfloor}^{r-1}\right) \oplus R_i\left(r\right), 0 \leqslant i \leqslant 23,\\
b_i^r &= b_i^{r-1} \oplus S_{i\,mod\,8}\left(B_{96}^{r-1}\right) \oplus R_i\left(r\right), 24 \leqslant i \leqslant 31,\\
b_i^r &= b_i^{r-1} \oplus b_{i-32}^r, 32 \leqslant i \leqslant 127.
\end{aligned}
\tag{6.1}
$$

where $1 \leqslant r \leqslant 10$, $R_i\left(r\right)$ is the i^{th} bit of the round constant word array $R\left(r\right)$, and $\lfloor x\rfloor$ is the floor function that returns the largest integer not greater than x. The relations among bits exhibited in Equation 6.1 show that each bit in the subsequent 10 rounds is

[1] A logical formula is considered to be in ANF if it is an XOR of a constant and one or more conjunctions of Boolean variables.

associated with its former bits. In particular, b_i^r $(0 \leqslant i \leqslant 31, 1 \leqslant r \leqslant 10)$ is determined by 9 bits, i.e., b_i^{r-1} and $B^{r-1}_{104+8\cdot\lfloor i/8 \rfloor}$, and b_i^r $(32 \leqslant i \leqslant 127, 1 \leqslant r \leqslant 10)$ is determined by 2 bits, i.e., b_i^{r-1} and b^r_{i-32}. Thus, an error occurring on a bit could be rectified by examining the values of its related 2 or 9 bits. Similarly, if several bits are decayed, they could be recovered correctly as long as a sufficient number of bits are known. For convenience, we call the relations characterized in Equation 6.1 as *bit-relations*. The bit-relations among two adjacent rounds are shown in Fig. 6.1, where the core refers to an S-box XORing the round-dependent word array $R(r)$.

6.4 SAT/ MaxSAT Encoding for Recovering AES-128 Key Schedules

The performance obtained in [49] confirmed the superiority of SAT solvers in AES key recovery. However, Abdel et al. [49] thoroughly excluded the possibility of reverse flipping errors, by assuming that all 1s are correct with absolute certainty. In fact, based on $\delta_1 = 0.1\%$, the event of reverse flipping is expected to arise around 1 or 2 times in a real key recovery attack for AES-128 where the total number of key bits is 1408, and more times for AES-192 and AES-256, which contain larger number of bits apt to decay. In particular, the reverse flipping errors arise 1 or 2 or 3 times in AES-192, and 2 or 3 or 4 times in AES-256. Although bits flipping from 0 to 1 are very rare in number, these fatal events are sufficient to derail SAT solvers, leading SAT solvers to mistake the wrong 1s as the correct constraints to infer the rest unknown bits. As a result, conflicts may occur during the reasoning and SAT solvers would fail to recover the correct key schedule by outputting unsatisfiable.

In this section, we show how to recover the AES key schedule with MaxSAT, an optimized version of SAT, in the presence of reverse flipping errors. Specifically, we extend the work of [49] to adapt SAT approach to a true cold boot attack. Moreover, motivated by the feature of MaxSAT which satisfies as many constraints as possible and eliminates the minority of unsatisfied ones, we recast the problem of AES key recovery under the realistic assumption as a partial MaxSAT problem.

6.4.1 Recovery with SAT under the Realistic Assumption

The versatility and effectiveness of SAT solving techniques have shown the potential use of SAT solvers as a tool for cryptanalysis. Several cryptanalytic attacks using SAT solvers emerged, ranging from the area of cryptanalysis of block ciphers [21, 22] and stream ciphers [27, 32] to asymmetric-key algorithms [26, 29, 79] and hash functions [40, 70]. Another line of research focuses on the attempts to make SAT solvers more cryptanalytic-friendly. Soos et al. [104] implemented several steps towards a specialized SAT solver for cryptography, including native support for the XOR operation, Gaussian elimination, and logical circuit generation. Facilitated by these XOR supported mechanisms in CryptoMiniSat, attacks against stream ciphers such as Crypto-1 [32] could be accelerated considerably, compared with other standard SAT solvers that only support CNF representations. The advantage of cryptanalytic-friendly SAT solvers is also demonstrated in [49], which fulfilled the AES key recovery with significantly less time than other previous works.

For a SAT solver, the attempt of finding an assignment to variables is made on the premise that all the specified constraints are satisfied without any exception, otherwise the solver outputs unsatifiable. This feature is suitable for the perfect assumption where all 1s present in the decayed AES key schedule are correct with absolute certainty. In addition, the bit-relations in the AES key schedule can be easily formulated as a SAT problem which lead itself naturally to SAT solvers [49]. Therefore, given enough number of 1s in the corrupted key schedule, a SAT solver could infer the values of other unknown bits by formulating all the 1s and bit-relations as hard constraints.

However, in case of some 0s flipping to 1s, a SAT solver could not be aware of such reverse flipping errors, instead, it still takes all 1s as absolutely correct. Misled by the incorrect information, the solver would fail to recover the key schedule by outputting unsatisfiable as a result of the conflicts arising during the reasoning. Obviously, the heart of addressing this problem is to find and rectify the reverse flipping errors. A straightforward way would be to conduct exhaustive search over all 1s in a bit-by-bit fashion, by exploiting the fact that a SAT solver would always output unsatisfiable as long as reverse flipping errors exist. Only if all the wrong 1s are cleared, the solver is probable to find the correct assignment of variables. Particularly, when there is one reverse flipping error, to distinguish this wrong 1 from other correct 1s, we modify the decayed key schedule by turning one of these 1s to 0, then take the modified key schedule as the input of a SAT solver. If the solver outputs satisfiable, it means that the bit that

we just modified is the reverse flipping error. Otherwise, we revert that bit; convert the next 1 to 0, and run the solver again. This process would repeat until the solver outputs satisfiable, i.e., it locates and corrects the reverse flipping error. In the worst case, the solver needs to run $(n+1)$ times, where n is the number of 1s in the decayed key schedule. The complexity of the exhaustive search is closely related to the actual number and the location of the reverse flipping errors, neither of which are unknown to a SAT solver. If the number of reverse flipping errors is k, in the best case, the solver needs to run $\left(\sum_{0\leqslant i\leqslant k-1} C_n^i + 1\right)$ times, while in the worst case, it has to try $\sum_{0\leqslant i\leqslant k} C_n^i$ times, where C_n^i is the number of i-combinations of n. It is clear that a SAT solver has to run multiple times to recover a decayed key schedule with reverse flipping errors. Obviously, this method is tedious and cumbersome.

6.4.2 Recovery with MaxSAT under the Realistic Assumption

We can solve the problem significantly better by taking advantage of the partial MaxSAT. That such a connection exists should be no surprise: we are in a situation where the majority of 1s are correct, with only a small fraction of 1s flipping from 0, and we wish to find out such reverse flipping errors and recover the true bits. A partial MaxSAT solver treats the bit-relations as hard constraints, while considers the bits equal to 1 as soft constraints, i.e., it takes the possibility that the present 1s may be incorrect into consideration. Solving the partial MaxSAT problem amounts to finding an assignment of variables that satisfy all hard constraints and the maximum number of soft constraints. As long as the reverse flipping errors account for a small percentage among all 1s, they can be surely cleared by a partial MaxSAT solver, which always satisfies the majority of soft constraints by excluding the unsatisfied minority. By using MaxSAT solvers, the exhaustive search over all 1s is eliminated.

Modeling 1s as soft clauses could be fulfilled without further elaboration. In particular, if the i^{th} bit of the r^{th} roundkey presents 1 in the decayed key schedule, we only need to declare $b_i^r = 1$ as a soft clause to a MaxSAT solver. By contrast, modeling bit-relations as hard clauses is not such straightforward. The main problem is to handle the XOR operation, which is fundamental in characterizing bit-relations. In the rest of this subsection, we elaborate the way of describing the following two kinds of formulas in CNF representations.

(1) XOR formula: a formula connected by XOR operator, with all the terms being individual Boolean variables.

(2) ANF formula: a formula connected by XOR operator, with terms consisting of a constant (0 or 1) and conjunctions of Boolean variables.

A naive way of describing an XOR formula in CNF representation introduces 2^{n-1} clauses, where n is the size of the XOR formula. This straightforward way, called *direct conversion*, applies to the case that the size of an XOR formula is relatively small.

Example 6.1. *In an AES-128 key schedule, to describe the bit-relation* $b_i^r = b_i^{r-1} \oplus b_{i-32}^{r-1}$ $(32 \leqslant i \leqslant 127, 1 \leqslant r \leqslant 10)$ *in CNF, only the following four clauses are sufficient:*

$$\neg b_i^r \vee b_i^{r-1} \vee b_{i-32}^{r-1}, \qquad b_i^r \vee \neg b_i^{r-1} \vee b_{i-32}^{r-1},$$
$$b_i^r \vee b_i^{r-1} \vee \neg b_{i-32}^{r-1}, \quad \neg b_i^r \vee \neg b_i^{r-1} \vee \neg b_{i-32}^{r-1}.$$

Obviously, this interpretation is unmanageable for large n as the number of clauses goes up exponentially with the size of an XOR formula. To overcome this exponential explosion, a long XOR formula is usually cut up into manageable groups, each represented by an auxiliary Boolean variable. In our work, we set the size of manageable groups as 5, i.e., we introduce an auxiliary variable for each 5 Boolean variables, thus the number of clauses to describe one group is no more than 2^5. The long XOR formula is then represented as a new XOR of $\lceil n/5 \rceil$ auxiliary variables, as well as a set of clauses for describing the manageable groups. If the size of the new XOR formula is still overlong, then more auxiliary Boolean variables are introduced to further cut up the formula. This process repeats until the size of the new XOR formula is manageable. We call this method as *cut-up conversion*.

Up to this point, we have discussed the way of describing both short and long XOR formulas in CNF representations. To encode ANF into CNF, we need to first introduce additional Boolean variables to substitute the conjunctions, so that turn the ANF formula to an XOR, which could be handled by either direct or cut-up conversion. Formally, a conjunction of Boolean variables, denoted by $x_0 \wedge x_1 \wedge \cdots \wedge x_n$, could be represented by a new Boolean variable a with $n+1$ additional clauses:

$$x_1 \vee \neg a,\ x_2 \vee \neg a, \ldots, x_n \vee \neg a, \neg x_1 \vee \neg x_2 \cdots \vee \neg x_n \vee a.$$

Thus the ANF formula is converted to a set of clauses and an XOR of a constant and several additional Boolean variables. If the constant is 0, we remove it safely without further conversion. Otherwise, we eliminate the constant 1 by turning one of the Boolean variables to its negation. Specifically, $1 \oplus x_1 \oplus x_2 \ \cdots \oplus x_n$ is tautologically equivalent to $\neg x_1 \oplus x_2 \ \cdots \oplus x_n$.

Example 6.2. *In an AES-128 key schedule, the* 129*-th bit, i.e.,* b_0^1*, is represented as* $b_0^1 = b_0^0 \oplus S_0\left(B_{104}^0\right) \oplus R_0\left(1\right)$*, where* $R_0\left(1\right) = 1$ *and* $S_0\left(B_{104}^0\right)$ *is an ANF function of 131 terms. This formula is equivalent to* $b_0^1 \oplus b_0^0 \oplus S_0\left(B_{104}^0\right) = 1$ *where the left side of the equal symbol is an ANF formula of 133 terms, made up of 6 individual Boolean variables and 127 conjunctions of two or more Boolean variables. Then we introduce 127 additional Boolean variables to substitute the conjunctions, thus turning the ANF formula to an XOR of 133 Boolean variables and 127 substituted equations. The 127 equations are then converted to a set of clauses that a MaxSAT solver could understand.*

To handle the long XOR formula of 133 Boolean variables, if we apply the direct conversion, the number of clauses is 2^{132}*, which is unmanageable for the capacity of modern computers. In the following, we employ the cut-up conversion to address the problem. Boolean variables in the long XOR formula are divided into* $\lceil 133/5 \rceil = 27$ *groups, each represented by an auxiliary Boolean variable. Thus the number of clauses introduced in this step is* $2^5 \cdot 26 + 2^3 = 840$*. The long XOR formula then degrades to a shorter one of 27 auxiliary variables and 840 clauses. Evidently, this new XOR formula is still too long to be converted into manageable CNF, which requires* 2^{26} *clauses. In the follow-up step, we make further efforts to cut the formula by introducing* $\lceil 27/5 \rceil = 6$ *auxiliary variables and* $2^5 \cdot 5 + 2^2 = 164$ *clauses. Finally, followed by the set of introduced clauses, we rewrite the formula as a new one of 6 variables, which could be converted directly to 32 clauses. To sum up, to describe the long XOR formula in CNF, the numbers of auxiliary variables and clauses are equal to* $27 + 6 = 33$ *and* $840 + 164 + 32 = 1036$*, respectively.*

Although the cut-up conversion for handling long XOR formulas is far from optimal, it does decrease the number of clauses for describing the S-box that is represented as eight ANF functions of more than 100 conjunctions of Boolean variables. By introducing auxiliary Boolean variables to cut up long XOR formulas, the total number of clauses for the S-box is strikingly reduced.

6.5 Experiment and Comparison

In this section, we experimentally evaluated the performance of AES-128 key recovery with SAT and MaxSAT, respectively, including generating problem instances, selecting solvers and comparing the results.

6.5.1 Generating Problem Instances

According to [34], in a true cold boot attack, δ_1 is around 0.1% and δ_0 increases as time goes on. To be consistent with the realistic assumption, we generate the problem instances as follows. The test generator first derives a key schedule from a randomly selected initial key where the number of 0s and 1s are approximately equal, then it randomly converts 1s to 0s with the probability of δ_0, and converts 0s to 1s with the probability of δ_1. We set $\delta_1 = 0.1\%$ and vary δ_0 from 30% to 76%. Since the time for recovering a key schedule observed in the experiments rises dramatically when δ_0 grows over 70%, we set δ_0 from 30% to 70% with 10% increments, and 70% to 76% with 2% increments, respectively. At each fixed decay factor δ_0, 100 problem instances are generated. In this setting, the number of reverse flipping errors in the generated problem instances ranges from 0 to 2.

To investigate the performances of SAT and MaxSAT approaches under specific number reverse flipping errors, we generate additional problem instances for the following cases:

(1) Each instance contains only 1 reverse flipping error.
(2) Each instance contains 2 reverse flipping errors.

The method of generating instances for case (1) is almost the same as that for the realistic assumption, except that we remove the setting of δ_1 and limit the number of reverse flipping errors to 1. For case (2), due to the extended time for key recovery, we range δ_0 from 30% to 74% and generate 40 instances for each fixed δ_0.

It is predictable that as δ_0 goes up, the number of "probably known" bits decreases. In this situation, SAT/ MaxSAT solvers need quite a long time to find the result, which may be different from the correct key schedule. Fortunately, in our experiments where δ_0 is up to 76%, the result for each problem instance obtained by the solvers is always unique, i.e., the same as the correct key schedule.

6.5.2 Selecting Appropriate Solvers

The solver for evaluating the SAT approach is CryptoMiniSat (version 2.9.6) as it supports XOR operations natively. The encodings of bit-relations are almost the same as [49], with the only difference on the strategy of handling an XOR formula like $x_0 \oplus x_1 \oplus x_2 = 0$. Considering that CryptoMiniSat expects only positive clauses, we rewrite the XOR formula as its equivalent form $\neg x_0 \oplus x_1 \oplus x_2 = 1$, instead of adding a new clause with a dummy variable d to turn the XOR formula to two formulas: $d \oplus x_0 \oplus x_1 \oplus x_2 = 1$ and $d = 1$, as illustrated in [49].

So far, there have been a variety of MaxSAT solvers, classified into two categories. The one implements a branch and bound scheme, and the other one uses a state-of-the-art SAT solver as an inference engine. Candidate partial MaxSAT solvers in our experiments are listed as follows: Akmaxsat (version 1.1) [53] and WMaxSatz (version 2009) [59] are branch and bound solvers. QMaxSAT (version 0.21) [52], QMaxSAT-g2 (version 0.21) [52], WPM1 (version 2012) [4], PM2 (version 2010) [4], and Pwbo (version 2.0) [67] are SAT-based solvers. To evaluate whether these solvers are suitable for AES key recovery, we first apply these candidates to recover 100 AES-128 key schedules with $\delta_1 = 0.1\%$ and $\delta_0 = 30\%$. Our experiment was carried out on a 2.2GHz quad-core Intel i7-2675 processor with 8GB RAM. Under a time limit of 900 seconds, WPM1, PM2, and Pwbo succeeded in recovering all the key schedules, while Akmaxsat and WMaxSatz accomplished none of these instances. In the middle are QMaxSAT and QMaxSAT-g2, which managed to solve 12 and 42 instances, respectively.

Next, we compare the performances of WPM1, PM2, and Pwbo with increasing decay factors. Specifically, δ_0 ranges from 30% to 72%, and δ_1 is set 0.1%. Results are illustrated in Table 6.1. Number in bracket means the number of instances that were successfully solved by the corresponding solver and is omitted in the table if the solver managed to recover all the 100 instances within the time limit. As can be seen from Table 6.1, within the time limit of 900 seconds, Pwbo managed to recover more key schedules than the other two solvers. In particular, when $\delta_0 = 70\%$, both Pwbo and PM2 recovered all the key schedules while WPM1 failed one of the instances. When $\delta_0 = 72\%$, Pwbo completed recovering 99 key schedules while the other two only fulfilled 97 instances. Moreover, among all the solved instances, the average runtime required for Pwbo is conspicuously shorter than that of WPM1 and PM2. Considering the high efficiency of Pwbo in recovering the AES key schedules, in the following experiments, we choose Pwbo as the solver to evaluate the MaxSAT approach.

TABLE 6.1: Average runtime (seconds) required for MaxSAT Solvers

δ_0 (%)	30	40	50	60	70	72
Pwbo	0.928	1.236	1.347	1.771	11.564	25.967 (99)
WPM1	1.601	2.195	2.771	4.140	25.993 (99)	41.659 (97)
PM2	1.060	1.541	2.778	4.205	22.114	48.666 (97)

TABLE 6.2: Average runtime of SAT/MaxSAT approaches at $\delta_1 = 0.1\%$

Decay Factor δ_0 (%)	CryptoMiniSat		Pwbo
	Solver Time (s)	CPU Time (s)	Solver Time (s)
30	45.812	462.238	0.943
40	28.467	228.878	0.956
50	19.665	97.240	1.168
60	26.524	89.187	1.560
70	225.379	255.097	12.532
72	678.452	718.031	26.782
74	1004.161	1035.134	231.610
76	1116.353	1143.289	296.415

6.5.3 Results

In the following experiments, we carried out tests on a 2.6GHz quad-core Intel i5-2540 processor with 8GB RAM and experimentally evaluated the performance of AES-128 key recovery with CryptoMiniSat and Pwbo, respectively. Both solvers were run on the same sets of test cases, so their results are directly comparable.

For the SAT approach, we collect two kinds of metrics: solver time and CPU time. The former one is the net time consumed by CryptoMiniSat, and the latter one is the total time spent in solving an instance, including the solver time as well as the time of modifying the input file, which is inevitable when searching for reverse flipping errors. Evidently, the CPU time is always longer than the solver time for solving the same set of instances. By contrast, we evaluate the MaxSAT approach by only measuring the solver time because this approach enables reliable key recovery by loading the input file only once. In what follows, we compare the performances of SAT and MaxSAT approaches with the solver time, and list the CPU time only for reference. The benchmark results for true cold boot attacks are summarized in Tab. 6.2.

In general, the MaxSAT approach enables more efficient key recovery than the SAT approach, at all the varied decay factors. Specifically, the solver time needed by the MaxSAT approach rises monotonically with the increase of δ_0, indicating the higher

difficulty in fulfilling the key recovery at increasing δ_0. In comparison, for the SAT approach, both the solver time and CPU time present a down and up trend as δ_0 increases. We explain the reasons as follows. First, at the low decay factor, particularly when δ_0 is 30%, the number of 1s in a decayed schedule is relatively large, implying that CryptoMiniSat needs to run a considerable number of times to find out the reverse flipping errors. At this point, both the two metrics are maintained in high level, especially the CPU time, because a large amount of time is spent in modifying the input file, with only a small fraction used for CryptoMiniSat to determine whether the input file is satisfiable. Later, with the increase of the decay factor, the number of 1s drops, leading to less time on modifying the input file and fewer number of times to run CryptoMiniSat. In addition, the increasing decay factor is not large enough to affect the high efficiency of CryptoMiniSat to solve an input file. In this situation, both the solver time and CPU time decline. Finally, as the decay factor further rises, the difficulty of solving an input file increases strikingly, and the net time consumed by CryptoMiniSat becomes the dominant factor that influences the total time. As a result, both metrics rise sharply when δ_0 is over 70%.

Table 6.2 demonstrates that the MaxSAT approach outperforms the SAT one in recovering AES-128 key schedules for a true cold boot attack under the realistic assumption, where the number of reverse flipping errors ranges from 0 to 2. Since the focus of this chapter is to handle problems with reverse flipping errors, in the following, we carried out two additional tests, as described in case (1) and (2), to estimate the performance of the two approaches under a specific number of reverse flipping errors. Table 6.3 and Tab. 6.4 reflect the runtime statistics of SAT and MaxSAT approaches with 1 reverse flipping error. Table 6.5 and Tab. 6.6 show the results with 2 reverse flipping errors.

When there is only one reverse flipping error, the MaxSAT approach runs slightly faster than the SAT approach, as indicated in Tab. 6.3 and Tab. 6.4. In particular, when the decay factor reaches 76%, the solver time of CryptoMiniSat for the worst case is more than 2.9 hours, with the average time at 8 minutes and median time at 3 minutes, respectively. In the MaxSAT approach, the worst case is recovered within 1.8 hours, with the average and median time at 6.4 and 2.4 minutes, respectively.

Table 6.5 and Tab. 6.6 show the time statistics of the SAT and MaxSAT approaches for the decay factor from 30% to 74%, with exactly 2 reverse flipping errors in each instance. Evidently, the MaxSAT approach is far superior to the other one at all decay factors. In particular, when the decay factor is 74%, solver time of CryptoMiniSat grows averagely to 4 hours with the median time at 2.2 hours. Moreover, nearly 2 days are consumed

Table 6.3: Runtime statistics of SAT/MaxSAT approaches with 1 reverse flipping error

Decay Factor δ_0 (%)			30	40	50	60	70
CryptoMiniSat	Solver Time(s)	Avg.	2.045	1.310	1.971	5.026	43.532
		Med.	1.761	1.169	1.443	2.285	29.016
		Max	5.161	3.942	8.911	78.318	642.389
		Min	0.090	0.070	0.111	0.180	1.045
		St.Dev.	1.423	0.967	1.632	8.678	69.660
	CPU Time(s)	Avg.	10.680	6.959	7.334	9.273	47.509
		Med.	9.141	6.378	6.024	6.240	31.804
		Max	23.133	18.029	20.877	88.638	646.952
		Min	0.328	0.164	0.296	0.224	1.108
		St.Dev.	6.647	5.009	5.157	10.578	70.630
Pwbo	Solver Time(s)	Avg.	1.037	1.088	1.345	2.252	14.945
		Med.	0.785	0.905	1.145	1.944	8.949
		Max	2.369	2.396	3.240	6.351	262.724
		Min	0.705	0.704	0.722	0.768	1.025
		St.Dev.	0.449	0.402	0.540	1.307	29.022

Table 6.4: Runtime statistics of SAT/MaxSAT approaches with 1 reverse flipping error (cont'd)

Decay Factor δ_0 (%)			72	74	76
CryptoMiniSat	Solver Time(s)	Avg.	47.603	280.825	480.101
		Med.	43.625	67.475	186.010
		Max	233.645	3491.271	10498.967
		Min	0.575	0.754	2.016
		St.Dev.	43.684	646.057	1196.448
	CPU Time(s)	Avg.	51.450	284.525	483.432
		Med.	37.295	71.431	188.626
		Max	240.951	5691.972	10503.143
		Min	1.256	0.788	2.616
		St.Dev.	45.064	750.881	1165.962
Pwbo	Solver Time(s)	Avg.	28.354	122.524	384.348
		Med.	13.311	25.224	142.748
		Max	599.011	3122.610	6492.834
		Min	1.514	2.278	4.335
		St.Dev.	61.904	344.664	869.453

Table 6.5: Runtime statistics of SAT/MaxSAT approaches with 2 reverse flipping errors

Decay Factor δ_0 (%)			30	40	50	60
CryptoMiniSat	Solver Time(s)	Avg.	198.638	162.249	224.689	329.621
		Med.	171.615	186.392	127.715	153.839
		Max	387.542	371.530	1358.530	2112.362
		Min	13.740	8.611	7.304	27.476
		St.Dev.	87.961	89.679	276.601	493.632
	CPU Time(s)	Avg.	1838.272	1438.002	1175.426	908.879
		Med.	1547.673	1582.850	902.144	884.411
		Max	3884.488	3182.162	4297.181	3362.834
		Min	64.041	37.678	65.888	68.824
		St.Dev.	1052.836	900.462	917.924	770.543
Pwbo	Solver Time(s)	Avg.	1.464	1.562	2.184	4.676
		Med.	1.020	1.165	1.884	3.165
		Max	7.121	8.218	5.834	19.192
		Min	0.229	0.901	0.910	1.002
		St.Dev.	1.114	1.223	1.197	3.942

Table 6.6: Runtime statistics of SAT/MaxSAT approaches with 2 reverse flipping errors (cont'd)

Decay Factor δ_0 (%)			70	72	74
CryptoMiniSat	Solver Time(s)	Avg.	3047.821	4909.565	14715.607
		Med.	2077.345	3517.530	7936.68
		Max	9747.162	17027.594	161389.012
		Min	38.417	10.439	62.768
		St.Dev.	2907.309	5077.264	26695.935
	CPU Time(s)	Avg.	3366.986	5309.834	14967.686
		Med.	2208.291	3663.322	8249.485
		Max	10226.602	17670.254	161885.498
		Min	145.917	29.246	90.738
		St.Dev.	3029.013	5224.465	26753.922
Pwbo	Solver Time(s)	Avg.	47.725	245.177	2160.486
		Med.	37.343	97.372	473.762
		Max	220.865	1848.418	20687.945
		Min	5.054	6.710	20.045
		St.Dev.	37.338	358.486	3982.245

by recovering the key schedule for the worst case. By contrast, the average and median time for Pwbo to solve the same set of instances is 36 and 7.9 minutes, respectively, with the worst-case recovery time at 5.7 hours. The low efficiency of the SAT approach is owing to the large number of times for searching the reverse flipping errors, while the MaxSAT solver only runs one time, by excluding the minority errors with optimized algorithms.

It is worth noting that when recovering the same decayed AES key schedule, the MaxSAT approach generates overwhelmingly larger number of clauses than the other one. To be specific, the number of hard clauses for capturing AES-128 bit-relations is 372,240 for the MaxSAT approach, around 6 times more than that for the SAT approach, which generates only 51,440 hard clauses. The conspicuous gap in numbers attributes to the distinct strategies for handling the XOR operation. As an XOR supported SAT solver, CryptoMiniSat prunes redundant clauses dynamically along with the process of assigning Boolean values to variables natively, considerably decreasing the number of clauses introduced for handling XOR formulas. By contrast, to date, there have not been any MaxSAT solvers that could support XOR natively, thus an XOR formula has to be converted to CNF by the direct conversion and cut-up conversion introduced in Section 6.4.2. Unfortunately, both conversions are executed in a static way, in spite that some introduced clauses are redundant for the current assignment of variables. Though the XOR-CNF conversions in MaxSAT are not as refined as the strategies of CryptoMiniSat, experiments still show that the MaxSAT approach excels the SAT one, and we say with assurance that the development of a MaxSAT solver that supports XOR natively would further stretch the advantage of the MaxSAT approach.

6.6 Chapter Summary

This chapter presented a MaxSAT approach to fulfill the reliable AES key recovery from a cold boot attack. In contrast to many existing solutions that take the perfect assumption for granted, our work enables efficient key recovery applicable to a true cold boot attack, as long as the reverse flipping errors account for a small fraction of all 1s. Specifically, we extended the work of [49] to make SAT solvers adapt to AES key recovery from a true cold boot attack. Moreover, motivated by the feature of MaxSAT that satisfies as many soft constraints as possible by eliminating the minority of unsatisfied ones, we recast the problem as a partial MaxSAT problem. Experiments showed that the MaxSAT approach outperforms the SAT approach, especially when the

number of reverse flipping bits is relatively large. In particular, with MaxSAT, decayed key schedules with $\delta_0 = 76\%$ and $\delta_1 = 0.1\%$ could be recovered within 300 seconds on average, while the SAT approach required nearly 4 times longer to solve the same set of instances.

Chapter 7

Conclusions and Future Works

This thesis has contributed to the research area of applying MaxSAT, an optimization version of the famous SAT problem, to multi-agent systems and cryptographic area. This chapter draws the conclusions from the research results obtained, and looks at some future work on MaxSAT research and related issues.

7.1 Efficient MaxSAT Encoding

Extended Weighted Partial MaxSAT. There is no argument that the most important feature of a MaxSAT solver is its efficiency. Therefore, a large number of applications that rely on a MaxSAT solver as a core to fulfill the task have achieved significant improvements compared to traditional optimization frameworks.

In order to facilitate the modelling of a variety of optimization problems, where some constraints may be more important to satisfy than others, the MaxSAT problem have been extended by associating different payoffs with satisfying different clauses. In this case, it is natural to cast MaxSAT in terms of maximizing the total payoffs of the satisfied clauses. One of the most general form of MaxSAT instances is commonly referred to as weighted partial MaxSAT (WPM). In WPM instances, each clause is associated with a weight that is either a positive integer or infinity. The solution to the WPM instance is a variable assignment that satisfies all the infinite weighted clauses and maximizes the total weights of satisfied clauses. WPM is more well-suited for representing and solving problems where satisfying some constraints is more crucial than others. The positive weight is used to measure the importance of constraints.

For some applications, the weights associated to constraints are ranging from negative to positive. For example, in the coalition structure generation (CSG) problem, to find the optimal partition of agents, different coalitions and the assigned payoffs need to be examined. In these coalitions, positive and negative payoffs may co-exist. In order to apply the off-the-shelf WPM solvers, converting weights from negative to positive is indispensable, as WPM solvers only deal with positive weights.

This thesis has described a formal method, called extend weighted partial MaxSAT (EWPM), to handle formulas with both positive and negative weights. Thus problems with non-zero weights can be readily formulated as EWPM instances. Next, we looked in the relationship between EWPM and WPM instances, presenting a procedure on EWPM-to-WPM transformation. With this transformation, EWPM instances are converted to standard WPM instances that are naturally solved by existing WPM solvers. Finally, as the result of the EWPM-to-WPM transformation, the ultimate results obtained by the WPM solvers may suffer from some deviations compared with the original ones. Therefore, the last step is to offset the effect of deviations by adding W_{neg}, the sum of negative weights in the problem.

MaxSAT Encoding for the CSG Problem. Coalition formation is a key research topic in multi-agent systems. One of the goals in coalition formation is to form a coalition structure that maximizes the sum of values of coalitions, known as the coalition structure generation (CSG) problem. Two critical problems for finding such coalition structure are:

1. the input size grows exponentially as the number of agents increases;
2. the number of possible coalition structures is too large to allow exhaustive search for the optimal one.

The first problem has been resolved by making use of compact representation schemes to describe the CSG problem. In these concise representations, the CSG problem is represented by a set of rules, in the form of $condition \rightarrow value$, where *condition* is denoted by conjunction (logic *and*) of agents, and *value* is a real-valued payoff. A rule is said *apply to* a coalition C if all agents that appear themselves in the rule are in C and none of agents that appear their negations in the rule are in C. Then the CSG problem evolves to find a coalition structure such that the sum of values of rules that apply to some coalitions in the coalition structure is maximized.

This thesis has taken the advantage of the compact representation schemes and developed two WPM encodings for the CSG problem, based on rule relations and agent relations respectively. The rule relation-based approach is directly encoded from the previous work [111]. Rule relations are classified into four exhaustive and non-overlapping categories. The main method is to identify the relations between each pair of rules and compare the payoffs of rule sets that apply to some coalitions under certain constraints. Finally the rule set that achieves the maximum payoff is the optimal solution. The agent relation-based approach identifies only two kinds of relations, i.e., whether two agents are in the same coalition or not, and thus makes the problem simpler compared to the other one. Some of the results obtained are: the rule relation-based WPM encoding is significantly time efficient than previous works, and the agent relation-based WPM encoding further improves the efficiency of the CSG problem solving, given the numbers of agents and rules are equal in a problem instance.

MaxSAT Encoding for Recovering AES Key Schedules. Advanced encryption standard (AES) is a worldwide prevalent symmetric cryptographic algorithm used to keep data confidential. An AES key refers to a key schedule made up of a series of 0-1 bits. This thesis has devised a partial MaxSAT encoding to recover the AES key schedule in the case that only a corrupted image of key bits are available. The image of key bits is especially extracted from the dynamic random access memory (DRAM), which persists for a short time after power is removed. An essential property of DRAM is that, after power is lost, most bits decay to the ground state, while a small fraction (only 0.1%) flips to the charged state. This thesis has made full use of this property and encoded the key recovery problem into partial MaxSAT, where all bits in the charged state are taken as soft constraints and the relations that have to be satisfied among key bits are taken as hard constraints. This encoding has achieved remarkable improvements compared to the previous works, especially when the phenomenon of "reverse flipping" is relatively remarkable.

7.2 Future Works

There are several avenues of research that we hope are eventually explored.

MaxSAT Encoding for Naive Representation of the CSG Problem. Naive representation of a characteristic function or a partition function is simply enumerating

all possible coalitions as well as the corresponding payoffs, thus requiring large representation size. A remedy for this problem is to represent the functions by compact representation schemes that significantly reduce the input size of the CSG problem.

In this thesis of solving the CSG problem with WPM encodings, no matter whether based on rule relations or agent relations, the problem is represented by a set of rules, thus a characteristic function or partition function is represented in more concise manner. Although these compact representation schemes have been shown expressive for coalitional games of which the size depends on the complexity of the interactions among agents, it remains unclear which kinds of problem domains can be concisely represented by such rules. It is likely that some problems are intractable for these concise representation schemes. In the future work, we would like to go back to the naive representation and try to apply MaxSAT solvers to solve the CSG problem expressed by the naive representation.

Handling XOR in MaxSAT. The potential use of SAT solvers as a tool for cryptanalysis has been proven in large number of cryptanalytic attacks, such as block ciphers, stream ciphers, and hash functions, etc. To date, attempts have been made to make SAT solvers more cryptanalytic-friendly by supporting the XOR operation natively, such as CryptoMiniSAT. This facilitates the application of SAT solvers in many cryptanalytic attacks, where XOR is the key operation in cryptographic algorithms.

Unfortunately, to date, there have not been any MaxSAT solvers able to handle XOR natively. This is prohibitive for applications in cryptographic areas where XOR dominates. That is why in Chapter 6, to recover the AES key schedules, XOR-CNF conversions are indispensable before a problem instances is input to a MaxSAT solver. In our current work, a long XOR formula is cut up into several shorter formulas. Each formula has up to six variables. This is far from optimal while works significantly better than SAT solvers. Therefore, it is promising if more advanced method of handling XOR formulas is devised for MaxSAT solvers. In the future work, we would like to refine the way of handling XOR formulas in MaxSAT, for example, seek the optimal value of k when cutting a long XOR formula into groups of k variables, and integrate the XOR supported mechanism into MaxSAT, so as to relieve the solver of the overwhelming large number of clauses.

Appendix A

Complete File for Example 4.7 in WPM Input Format

The following shows the generated file for solving Example 4.7 in WPM input format.

```
c #agent : 4 #rule : 6
c the maximum cost : 5
p wcnf 21 47 6
c
c soft clause
2 1 0
1 2 0
1 3 0
1 4 0
c
c hard clauses generated from positve rules with ex
6 -4 5 0
6 -4 6 0
c
c hard clauses generated from compS
6 -1 -5 10 0
6 -10 1 0
6 -10 5 0
6 -3 -6 18 0
6 -18 3 0
```

6 -18 6 0
c
c hard clauses generated from compD
6 -1 -3 -8 0
6 -2 -3 -12 0
6 -2 -6 -15 0
6 -3 -5 -17 0
6 -5 -6 -21 0
c
c hard clauses generated from incomp
c
c transitive laws
6 -7 -12 8 0
6 -7 -14 10 0
6 -7 -15 11 0
6 -7 -8 12 0
6 -7 -10 14 0
6 -7 -11 15 0
6 -8 -12 7 0
6 -10 -14 7 0
6 -11 -15 7 0
6 -8 -17 10 0
6 -8 -18 11 0
6 -8 -10 17 0
6 -8 -11 18 0
6 -10 -17 8 0
6 -11 -18 8 0
6 -10 -21 11 0
6 -10 -11 21 0
6 -11 -21 10 0
6 -12 -17 14 0
6 -12 -18 15 0
6 -12 -14 17 0
6 -12 -15 18 0
6 -14 -17 12 0
6 -15 -18 12 0
6 -14 -21 15 0
6 -14 -15 21 0

6 -15 -21 14 0
6 -17 -21 18 0
6 -17 -18 21 0
6 -18 -21 17 0

Appendix B

Complete File for Example 4.8 in WPM Input Format

The following shows the generated file for solving Example 4.8 in WPM input format.

```
c #agent : 4 #rule : 8
c the maximum cost : 3
p wcnf 36 143 6
c
c soft clause
2 1 0
1 2 0
1 -3 0
1 1-4 0
c
c hard clauses generated from dummy rules without ex
6 3 5 0
c
c hard clauses generated from negative rules with ex
6 4 6 7 8 0
c
c hard clauses generated from compS
6 -1 -5 12 0
6 -12 1 0
6 -12 5 0
```

```
6 -1 -7 14 0
6 -14 1 0
6 -14 7 0
6 -3 -8 26 0
6 -26 3 0
6 -26 8 0
6 -5 -6 31 0
6 -31 5 0
6 -31 6 0
6 -5 -7 32 0
6 -32 5 0
6 -32 7 0
6 -5 -8 33 0
6 -33 5 0
6 -33 8 0
6 -6 -7 34 0
6 -34 6 0
6 -34 7 0
6 -7 -8 36 0
6 -36 7 0
6 -36 8 0
c
c hard clauses generated from compD
6 -1 -3 -10 0
6 -2 -3 -16 0
6 -2 -5 -18 0
6 -2 -7 -20 0
6 -3 -6 -24 0
c
c hard clauses generated from incomp
6 -1 -6 0
6 -2 -8 0
6 -3 -7 0
c
c transitive laws
6 -9 -16 10 0
6 -9 -18 12 0
6 -9 -19 13 0
```

6 -9 -20 14 0
6 -9 -21 15 0
6 -9 -10 16 0
6 -9 -12 18 0
6 -9 -13 19 0
6 -9 -14 20 0
6 -9 -15 21 0
6 -10 -16 9 0
6 -12 -18 9 0
6 -13 -19 9 0
6 -14 -20 9 0
6 -15 -21 9 0
6 -10 -23 12 0
6 -10 -24 13 0
6 -10 -25 14 0
6 -10 -26 15 0
6 -10 -12 23 0
6 -10 -13 24 0
6 -10 -14 25 0
6 -10 -15 26 0
6 -12 -23 10 0
6 -13 -24 10 0
6 -14 -25 10 0
6 -15 -26 10 0
6 -12 -31 13 0
6 -12 -32 14 0
6 -12 -33 15 0
6 -12 -13 31 0
6 -12 -14 32 0
6 -12 -15 33 0
6 -13 -31 12 0
6 -14 -32 12 0
6 -15 -33 12 0
6 -13 -34 14 0
6 -13 -35 15 0
6 -13 -14 34 0
6 -13 -15 35 0
6 -14 -34 13 0

6 -15 -35 13 0
6 -14 -36 15 0
6 -14 -15 36 0
6 -15 -36 14 0
6 -16 -23 18 0
6 -16 -24 19 0
6 -16 -25 20 0
6 -16 -26 21 0
6 -16 -18 23 0
6 -16 -19 24 0
6 -16 -20 25 0
6 -16 -21 26 0
6 -18 -23 16 0
6 -19 -24 16 0
6 -20 -25 16 0
6 -21 -26 16 0
6 -18 -31 19 0
6 -18 -32 20 0
6 -18 -33 21 0
6 -18 -19 31 0
6 -18 -20 32 0
6 -18 -21 33 0
6 -19 -31 18 0
6 -20 -32 18 0
6 -21 -33 18 0
6 -19 -34 20 0
6 -19 -35 21 0
6 -19 -20 34 0
6 -19 -21 35 0
6 -20 -34 19 0
6 -21 -35 19 0
6 -20 -36 21 0
6 -20 -21 36 0
6 -21 -36 20 0
6 -23 -31 24 0
6 -23 -32 25 0
6 -23 -33 26 0
6 -23 -24 31 0

6 -23 -25 32 0
6 -23 -26 33 0
6 -24 -31 23 0
6 -25 -32 23 0
6 -26 -33 23 0
6 -24 -34 25 0
6 -24 -35 26 0
6 -24 -25 34 0
6 -24 -26 35 0
6 -25 -34 24 0
6 -26 -35 24 0
6 -25 -36 26 0
6 -25 -26 36 0
6 -26 -36 25 0
6 -31 -34 32 0
6 -31 -35 33 0
6 -31 -32 34 0
6 -31 -33 35 0
6 -32 -34 31 0
6 -33 -35 31 0
6 -32 -36 33 0
6 -32 -33 36 0
6 -33 -36 32 0
6 -34 -36 35 0
6 -34 -35 36 0
6 -35 -36 34 0

Appendix C

Complete File for Example 5.2 in WPM Input Format

The following shows the generated file for solving Example 5.2 in WPM input format.

```
c #agent : 4 #rule : 4
c the maximum cost : 5
p wcnf 10 22 6
c
c soft clauses and their derived hard clauses
2 7 0
6 -7 1 0
c
1 8 0
6 -8 -6 0
c
1 9 0
6 -9 -3 0
c
1 10 0
6 -10 1 0
6 -10 -3 0
6 -10 -6 0
c
c transitive laws
```

```
6 -1 -2 4 0
6 -1 -3 5 0
6 -1 -4 2 0
6 -1 -5 3 0
6 -2 -3 6 0
6 -2 -6 3 0
6 -2 -4 1 0
6 -3 -5 1 0
6 -3 -6 2 0
6 -4 -5 6 0
6 -4 -6 5 0
6 -5 -6 4 0
```

Appendix D

Complete File for Example 5.3 in WPM Input Format

The following shows the generated file for solving Example 5.3 in WPM input format.

```
c #agent : 4 #rule : 4
c the maximum cost : 3
p wcnf 10 20 6
c
c soft clauses and their derived hard clauses
2 7 0
6 -7 1 0
c
1 8 0
6 -8 -6 0
c
1 -9 0
6 9 3 0
c
1 -10 0
6 10 -1 3 2 0
c
c transitive laws
6 -1 -2 4 0
6 -1 -3 5 0
```

```
6 -1 -4 2 0
6 -1 -5 3 0
6 -2 -3 6 0
6 -2 -6 3 0
6 -2 -4 1 0
6 -3 -5 1 0
6 -3 -6 2 0
6 -4 -5 6 0
6 -4 -6 5 0
6 -5 -6 4 0
```

Appendix E

A Sample of Sbox Expressed in ANF

Let $B_0 = \{b_0, b_1, b_2, b_3, b_4, b_5, b_6, b_7\}$, $b_0b_1b_2b_4b_5b_6b_7$ be short for $b_0 \wedge b_1b_2 \wedge b_4 \wedge b_5 \wedge b_6 \wedge b_7$, then $S_i\,(B_0)$ $(i = 0, \ldots, 7)$ is represented as follows.

$$
\begin{aligned}
S_0\,(B_0) = {} & b_0b_1b_2b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_4b_6b_7 \oplus b_0b_2b_3b_4b_5b_6b_7 \oplus b_0b_2b_3b_4b_5b_6 \oplus \\
& b_1b_2b_3b_4b_6b_7 \oplus b_0b_1b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_5b_7 \oplus b_0b_1b_2b_3b_6b_7 \oplus b_1b_3b_4b_5b_6b_7 \oplus \\
& b_0b_1b_2b_4b_5b_7 \oplus b_0b_1b_2b_3b_4b_7 \oplus b_0b_1b_3b_4b_6b_7 \oplus b_0b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_5b_6b_7 \oplus \\
& b_2b_3b_4b_5b_6b_7 \oplus b_1b_2b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_4b_6 \oplus b_0b_2b_3b_4b_5b_7 \oplus b_0b_3b_5b_6b_7 \oplus b_0b_2b_3b_4b_5 \oplus \\
& b_2b_3b_4b_5b_7 \oplus b_1b_2b_4b_6b_7 \oplus b_0b_1b_5b_6b_7 \oplus b_0b_3b_4b_5b_7 \oplus b_0b_1b_2b_3b_6 \oplus b_0b_1b_3b_5b_7 \oplus \\
& b_0b_1b_2b_6b_7 \oplus b_0b_2b_4b_6b_7 \oplus b_1b_2b_4b_5b_6 \oplus b_0b_1b_4b_5b_6 \oplus b_0b_1b_2b_4b_7 \oplus b_0b_2b_3b_4b_6 \oplus \\
& b_0b_1b_2b_3b_4 \oplus b_0b_1b_2b_4b_5 \oplus b_1b_2b_5b_6b_7 \oplus b_0b_1b_3b_5b_6 \oplus b_1b_2b_3b_5b_7 \oplus b_1b_3b_4b_5b_7 \oplus \\
& b_0b_3b_4b_5b_6 \oplus b_0b_3b_4b_6b_7 \oplus b_1b_2b_3b_5b_6 \oplus b_0b_1b_2b_5b_7 \oplus b_0b_1b_3b_4b_6 \oplus b_2b_3b_4b_5b_6 \oplus \\
& b_0b_1b_3b_6b_7 \oplus b_2b_4b_5b_6b_7 \oplus b_0b_2b_4b_5b_7 \oplus b_0b_1b_2b_3b_7 \oplus b_0b_2b_5b_7 \oplus b_1b_2b_4b_6 \oplus b_0b_1b_2b_7 \oplus \\
& b_3b_5b_6b_7 \oplus b_1b_3b_6b_7 \oplus b_0b_2b_4b_5 \oplus b_0b_4b_5b_6 \oplus b_1b_2b_3b_5 \oplus b_0b_1b_2b_3 \oplus b_0b_2b_3b_5 \oplus \\
& b_2b_3b_5b_7 \oplus b_0b_1b_4b_7 \oplus b_0b_1b_4b_5 \oplus b_1b_3b_4b_6 \oplus b_2b_4b_5b_6 \oplus b_0b_1b_2b_5 \oplus b_2b_4b_5b_7 \oplus \\
& b_2b_3b_6b_7 \oplus b_1b_2b_3b_6 \oplus b_0b_4b_5b_7 \oplus b_1b_5b_6b_7 \oplus b_1b_2b_4b_7 \oplus b_0b_3b_5b_6 \oplus b_0b_2b_3b_7 \oplus \\
& b_1b_2b_6b_7 \oplus b_1b_2b_3b_7 \oplus b_1b_4b_5b_7 \oplus b_0b_4b_6b_7 \oplus b_0b_3b_4b_6 \oplus b_1b_4b_5b_6 \oplus b_0b_1b_2b_6 \oplus b_0b_2b_5b_6 \oplus \\
& b_0b_2b_4b_7 \oplus b_1b_2b_3 \oplus b_3b_5b_7 \oplus b_0b_2b_4 \oplus b_3b_4b_7 \oplus b_1b_2b_6 \oplus b_1b_4b_6 \oplus b_0b_4b_6 \oplus b_0b_1b_6 \oplus \\
& b_1b_3b_7 \oplus b_0b_3b_5 \oplus b_2b_3b_5 \oplus b_2b_5b_7 \oplus b_2b_3b_7 \oplus b_0b_3b_6 \oplus b_0b_1b_4 \oplus b_5b_6b_7 \oplus b_1b_2b_4 \oplus \\
& b_0b_4b_7 \oplus b_0b_2b_5 \oplus b_0b_1b_7 \oplus b_0b_3b_4 \oplus b_2b_5b_6 \oplus b_2b_6b_7 \oplus b_0b_2b_7 \oplus b_4b_5b_6 \oplus b_3b_6b_7 \oplus \\
& b_1b_3b_4 \oplus b_1b_5b_6 \oplus b_0b_2b_6 \oplus b_2b_4b_7 \oplus b_1b_4 \oplus b_1b_6 \oplus b_0b_6 \oplus b_0b_1 \oplus b_0b_5 \oplus b_6b_7 \oplus b_2b_7 \oplus \\
& b_2b_6 \oplus b_2b_4 \oplus b_5b_6 \oplus b_1b_3 \oplus b_1b_2 \oplus b_0b_4 \oplus b_2b_3 \oplus b_4b_6 \oplus b_5b_7 \oplus b_0 \oplus b_2 \oplus b_3 \oplus \neg b_4.
\end{aligned}
$$

$$S_1(B_0) = b_0b_1b_2b_3b_5b_6b_7 \oplus \oplus b_0b_1b_3b_4b_5b_6b_7 \oplus b_0b_1b_2b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_4b_6b_7 \oplus b_0b_2b_3b_4b_6b_7 \oplus b_0b_1b_3b_4b_6b_7 \oplus b_1b_3b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_5b_6 \oplus b_1b_2b_3b_4b_6b_7 \oplus b_0b_1b_3b_4b_5b_6 \oplus b_0b_1b_2b_4b_6b_7 \oplus b_0b_1b_2b_4b_5b_7 \oplus b_1b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_4b_6 \oplus b_1b_2b_3b_4b_5b_6 \oplus b_0b_2b_3b_5b_6b_7 \oplus b_0b_2b_3b_4b_5b_7 \oplus b_0b_1b_2b_3b_4b_7 \oplus b_0b_1b_2b_5b_6b_7 \oplus b_0b_2b_3b_4b_5b_6 \oplus b_1b_3b_4b_5b_6 \oplus b_1b_2b_3b_5b_6 \oplus b_0b_1b_2b_3b_6 \oplus b_0b_2b_5b_6b_7 \oplus b_2b_3b_4b_5b_7 \oplus b_0b_1b_3b_4b_7 \oplus b_1b_2b_3b_4b_6 \oplus b_0b_2b_3b_5b_7 \oplus b_0b_1b_2b_6b_7 \oplus b_0b_1b_2b_3b_4 \oplus b_0b_1b_3b_4b_6 \oplus b_0b_2b_4b_5b_6 \oplus b_0b_3b_5b_6b_7 \oplus b_0b_1b_5b_6b_7 \oplus b_0b_1b_2b_4b_7 \oplus b_0b_1b_2b_3b_7 \oplus b_0b_1b_3b_5b_7 \oplus b_1b_2b_4b_5b_6 \oplus b_0b_2b_3b_6b_7 \oplus b_2b_3b_4b_6b_7 \oplus b_0b_2b_3b_4b_7 \oplus b_0b_3b_4b_5b_7 \oplus b_0b_1b_2b_4b_6 \oplus b_1b_3b_4b_5b_7 \oplus b_0b_1b_3b_6b_7 \oplus b_0b_1b_2b_4b_5 \oplus b_0b_1b_4b_5b_6 \oplus b_0b_1b_3b_5b_6 \oplus b_0b_4b_5b_6b_7 \oplus b_2b_4b_5b_6b_7 \oplus b_0b_2b_3b_4b_5 \oplus b_0b_1b_4b_7 \oplus b_1b_3b_4b_7 \oplus b_0b_4b_5b_7 \oplus b_0b_2b_5b_6 \oplus b_0b_3b_4b_6 \oplus b_1b_2b_6b_7 \oplus b_1b_3b_4b_5 \oplus b_1b_3b_5b_7 \oplus b_1b_2b_5b_7 \oplus b_2b_3b_5b_7 \oplus b_0b_3b_4b_5 \oplus b_2b_3b_4b_5 \oplus b_1b_2b_3b_5 \oplus b_0b_1b_5b_6 \oplus b_2b_4b_5b_7 \oplus b_0b_1b_2b_6 \oplus b_1b_4b_6b_7 \oplus b_0b_4b_5b_6 \oplus b_0b_1b_3b_6 \oplus b_3b_4b_6b_7 \oplus b_0b_2b_4b_5 \oplus b_1b_2b_4b_7 \oplus b_0b_4b_6b_7 \oplus b_3b_4b_5b_6 \oplus b_1b_3b_5b_6 \oplus b_1b_4b_5b_7 \oplus b_0b_2b_3b_6 \oplus b_3b_4b_5b_7 \oplus b_0b_1b_4b_5 \oplus b_0b_1b_3b_4 \oplus b_2b_3b_4b_7 \oplus b_2b_4b_6b_7 \oplus b_1b_2b_5b_6 \oplus b_3b_5b_6 \oplus b_1b_3b_6 \oplus b_2b_3b_5 \oplus b_0b_5b_7 \oplus b_2b_6b_7 \oplus b_2b_3b_6 \oplus b_1b_3b_4 \oplus b_5b_6b_7 \oplus b_3b_6b_7 \oplus b_1b_2b_4 \oplus b_0b_3b_4 \oplus b_0b_2b_3 \oplus b_3b_5b_7 \oplus b_3b_4b_7 \oplus b_0b_2b_7 \oplus b_3b_4b_6 \oplus b_0b_1b_4 \oplus b_1b_2b_3 \oplus b_0b_1b_6 \oplus b_0b_6b_7 \oplus b_1b_3b_7 \oplus b_0b_4b_6 \oplus b_2b_5b_7 \oplus b_2b_3b_4 \oplus b_1b_3b_5 \oplus b_1b_4b_7 \oplus b_0b_4b_7 \oplus b_0b_1b_3 \oplus b_1b_5b_6 \oplus b_0b_4b_5 \oplus b_0b_4 \oplus b_3b_7 \oplus b_0b_7 \oplus b_0b_1 \oplus b_2b_3 \oplus b_4b_5 \oplus b_1b_3 \oplus b_2b_7 \oplus b_0b_3 \oplus b_4b_6 \oplus b_1b_7 \oplus b_2b_6 \oplus b_1b_4 \oplus b_0b_2 \oplus b_7 \oplus b_6 \oplus b_3 \oplus \neg b_0.$$

$$S_2(B_0) = b_0b_2b_3b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_4b_6b_7 \oplus b_0b_1b_3b_4b_5b_6b_7 \oplus b_1b_2b_3b_4b_5b_6b_7 \oplus b_0b_2b_3b_5b_6b_7 \oplus b_1b_2b_3b_4b_5b_7 \oplus b_0b_1b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_4b_5 \oplus b_1b_2b_3b_4b_6b_7 \oplus b_0b_1b_2b_3b_5b_6 \oplus b_0b_1b_3b_4b_6b_7 \oplus b_0b_1b_2b_4b_5b_7 \oplus b_0b_1b_2b_3b_4b_6 \oplus b_0b_1b_2b_3b_4b_7 \oplus b_0b_2b_3b_4b_5b_6 \oplus b_0b_1b_2b_4b_5b_6 \oplus b_2b_3b_4b_5b_6b_7 \oplus b_0b_1b_4b_5b_6b_7 \oplus b_0b_2b_3b_4b_6b_7 \oplus b_0b_2b_4b_5b_6b_7 \oplus b_0b_1b_3b_4b_5b_7 \oplus b_0b_3b_4b_5b_7 \oplus b_1b_3b_4b_6b_7 \oplus b_0b_1b_2b_6b_7 \oplus b_0b_1b_4b_6b_7 \oplus b_1b_4b_5b_6b_7 \oplus b_1b_2b_4b_5b_6 \oplus b_0b_2b_3b_4b_7 \oplus b_3b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_4 \oplus b_1b_2b_3b_4b_6 \oplus b_1b_3b_5b_6b_7 \oplus b_0b_2b_3b_5b_7 \oplus b_1b_2b_3b_4b_5 \oplus b_1b_2b_4b_5b_7 \oplus b_1b_3b_4b_5b_6 \oplus b_0b_1b_3b_4b_7 \oplus b_0b_2b_5b_6b_7 \oplus b_1b_2b_4b_6b_7 \oplus b_0b_1b_2b_3b_6 \oplus b_0b_1b_3b_6b_7 \oplus b_0b_1b_3b_4b_6 \oplus b_0b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_5 \oplus b_0b_1b_2b_3b_7 \oplus b_0b_3b_5b_6b_7 \oplus b_1b_2b_3b_6b_7 \oplus b_0b_2b_4b_5b_6 \oplus b_0b_1b_2b_5b_7 \oplus b_2b_3b_5b_6b_7 \oplus b_2b_3b_4b_6b_7 \oplus b_0b_1b_2b_4b_7 \oplus b_0b_1b_2b_4b_5 \oplus b_0b_1b_3b_5b_7 \oplus b_0b_2b_3b_4 \oplus b_3b_5b_6b_7 \oplus b_1b_2b_3b_5 \oplus b_1b_2b_5b_7 \oplus b_1b_3b_5b_7 \oplus b_0b_3b_5b_6 \oplus b_2b_4b_5b_7 \oplus b_0b_1b_3b_4 \oplus b_1b_5b_6b_7 \oplus b_0b_3b_4b_7 \oplus b_3b_4b_5b_6 \oplus b_0b_2b_3b_7 \oplus b_0b_1b_2b_5 \oplus b_2b_4b_6b_7 \oplus b_0b_1b_3b_5 \oplus b_0b_2b_4b_7 \oplus b_0b_1b_2b_7 \oplus b_0b_1b_5b_6 \oplus b_2b_5b_6b_7 \oplus b_4b_5b_6b_7 \oplus b_0b_3b_4b_6 \oplus b_1b_2b_6b_7 \oplus b_2b_3b_4b_7 \oplus b_1b_2b_3b_4 \oplus b_0b_1b_3b_6 \oplus b_0b_2b_3b_5 \oplus b_1b_2b_3b_6 \oplus b_0b_1b_4b_6 \oplus b_1b_4b_5b_7 \oplus b_2b_4b_5b_6 \oplus b_2b_3b_5b_6 \oplus b_0b_3b_5b_7 \oplus b_1b_3b_6b_7 \oplus b_1b_2b_4b_7 \oplus b_0b_2b_4b_6 \oplus b_2b_3b_4b_6 \oplus b_0b_4b_6 \oplus b_2b_3b_4 \oplus b_0b_3b_6 \oplus b_1b_2b_5 \oplus b_0b_3b_7 \oplus b_3b_5b_6 \oplus b_0b_2b_7 \oplus b_3b_4b_5 \oplus b_0b_1b_4 \oplus b_0b_6b_7 \oplus b_2b_4b_5 \oplus b_2b_3b_7 \oplus b_1b_2b_7 \oplus b_1b_5b_7 \oplus b_0b_2b_5 \oplus b_0b_2b_4 \oplus b_0b_1b_7 \oplus b_1b_3b_5 \oplus b_1b_3b_4 \oplus b_1b_4b_5 \oplus b_1b_2b_6 \oplus b_0b_3b_4 \oplus$$

$b_4b_6b_7 \oplus b_0b_5b_7 \oplus b_0b_4b_7 \oplus b_1b_2b_4 \oplus b_0b_1b_3 \oplus b_0b_1b_5 \oplus b_1b_4b_7 \oplus b_3b_4b_6 \oplus b_1b_3b_7 \oplus b_0b_3b_5 \oplus b_2b_6b_7 \oplus b_0b_2b_3 \oplus b_2b_3b_6 \oplus b_1b_5b_6 \oplus b_0b_4b_5 \oplus b_0b_1b_6 \oplus b_5b_6 \oplus b_0b_2 \oplus b_0b_4 \oplus b_1b_4 \oplus b_6b_7 \oplus b_0b_5 \oplus b_2b_3 \oplus b_4b_7 \oplus b_0b_7 \oplus b_0b_6 \oplus b_3b_4 \oplus b_0b_3 \oplus b_7 \oplus b_0 \oplus b_1 \oplus b_5.$

$S_3(B_0) = b_0b_1b_2b_3b_4b_5b_6 \oplus b_0b_1b_3b_4b_5b_6b_7 \oplus b_0b_2b_3b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_6b_7 \oplus b_0b_1b_3b_4b_5b_7 \oplus b_0b_1b_2b_3b_4b_6 \oplus b_1b_2b_3b_4b_5b_6 \oplus b_0b_1b_2b_4b_5b_7 \oplus b_1b_2b_3b_4b_5b_7 \oplus b_2b_3b_4b_5b_6b_7 \oplus b_1b_2b_4b_5b_6b_7 \oplus b_0b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_5b_7 \oplus b_0b_1b_2b_3b_5b_6 \oplus b_1b_2b_3b_4b_6b_7 \oplus b_0b_1b_2b_4b_6b_7 \oplus b_1b_2b_3b_5b_6b_7 \oplus b_0b_2b_3b_4b_6b_7 \oplus b_0b_1b_2b_3b_4b_7 \oplus b_0b_3b_4b_5b_6 \oplus b_0b_2b_3b_5b_7 \oplus b_0b_2b_4b_5b_7 \oplus b_2b_3b_4b_5b_6 \oplus b_0b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_5 \oplus b_0b_3b_4b_6b_7 \oplus b_0b_2b_5b_6b_7 \oplus b_0b_1b_2b_5b_6 \oplus b_2b_4b_5b_6b_7 \oplus b_0b_1b_2b_4b_7 \oplus b_0b_1b_5b_6b_7 \oplus b_0b_1b_2b_3b_4 \oplus b_0b_1b_2b_3b_7 \oplus b_3b_4b_5b_6b_7 \oplus b_0b_1b_4b_5b_7 \oplus b_0b_2b_4b_5b_6 \oplus b_0b_2b_3b_4b_5 \oplus b_1b_3b_4b_6b_7 \oplus b_0b_1b_3b_4b_7 \oplus b_1b_2b_3b_5b_6 \oplus b_0b_1b_2b_5b_7 \oplus b_2b_3b_4b_5b_7 \oplus b_0b_1b_3b_4b_5 \oplus b_0b_1b_3b_5b_6 \oplus b_1b_2b_4b_5b_6 \oplus b_0b_1b_3b_4b_6 \oplus b_1b_2b_5b_6b_7 \oplus b_0b_3b_4b_5b_7 \oplus b_0b_2b_3b_5b_6 \oplus b_0b_3b_5b_6b_7 \oplus b_1b_3b_5b_6b_7 \oplus b_0b_2b_3b_6b_7 \oplus b_0b_3b_4b_6 \oplus b_0b_1b_2b_3 \oplus b_0b_4b_5b_6 \oplus b_0b_2b_6b_7 \oplus b_1b_2b_6b_7 \oplus b_1b_3b_4b_6 \oplus b_4b_5b_6b_7 \oplus b_1b_2b_4b_5 \oplus b_1b_4b_5b_7 \oplus b_1b_2b_3b_7 \oplus b_1b_3b_5b_6 \oplus b_2b_4b_5b_7 \oplus b_2b_3b_4b_5 \oplus b_1b_2b_3b_6 \oplus b_1b_4b_5b_6 \oplus b_1b_3b_4b_5 \oplus b_3b_4b_6b_7 \oplus b_1b_5b_6b_7 \oplus b_0b_5b_6b_7 \oplus b_0b_1b_4b_6 \oplus b_1b_4b_6b_7 \oplus b_2b_4b_5b_6 \oplus b_0b_1b_2b_4 \oplus b_1b_2b_3b_5 \oplus b_0b_1b_3b_6 \oplus b_3b_4b_5b_6 \oplus b_0b_1b_2b_5 \oplus b_0b_2b_4b_6 \oplus b_0b_1b_5b_6 \oplus b_1b_2b_4b_7 \oplus b_0b_2b_3b_7 \oplus b_1b_3b_4b_7 \oplus b_1b_2b_5b_7 \oplus b_3b_5b_6b_7 \oplus b_2b_4b_6b_7 \oplus b_0b_1b_6b_7 \oplus b_0b_2b_5b_6 \oplus b_0b_2b_4b_7 \oplus b_0b_3b_6b_7 \oplus b_0b_2b_7 \oplus b_0b_4b_5 \oplus b_2b_3b_7 \oplus b_0b_2b_3 \oplus b_0b_1b_7 \oplus b_0b_1b_3 \oplus b_0b_2b_6 \oplus b_0b_3b_7 \oplus b_0b_3b_4 \oplus b_0b_3b_6 \oplus b_0b_4b_6 \oplus b_3b_4b_7 \oplus b_1b_2b_6 \oplus b_0b_1b_4 \oplus b_1b_2b_7 \oplus b_1b_2b_5 \oplus b_2b_4b_5 \oplus b_2b_3b_4 \oplus b_2b_5b_7 \oplus b_2b_3b_5 \oplus b_0b_1b_6 \oplus b_1b_3b_4 \oplus b_0b_1b_5 \oplus b_1b_5b_6 \oplus b_0b_2b_4 \oplus b_5b_6b_7 \oplus b_3b_5b_6 \oplus b_1b_2b_3 \oplus b_3b_5b_7 \oplus b_6b_7 \oplus b_2b_3 \oplus b_0b_3 \oplus b_5b_6 \oplus b_0b_7 \oplus b_3b_7 \oplus b_4b_5 \oplus b_1b_7 \oplus b_1b_2 \oplus b_5b_7 \oplus b_3b_6 \oplus b_0 \oplus b_6 \oplus b_4 \oplus b_7.$

$S_4(B_0) = b_0b_1b_2b_3b_4b_5b_7 \oplus b_0b_1b_2b_3b_4b_6b_7 \oplus b_0b_2b_3b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_4b_5 \oplus b_1b_2b_3b_4b_5b_6 \oplus b_0b_1b_2b_3b_4b_7 \oplus b_0b_1b_2b_3b_5b_6 \oplus b_1b_2b_3b_5b_6b_7 \oplus b_0b_3b_4b_5b_6b_7 \oplus b_1b_2b_3b_4b_6b_7 \oplus b_1b_2b_3b_4b_5b_7 \oplus b_0b_1b_3b_4b_5b_6 \oplus b_0b_1b_2b_4b_5b_6 \oplus b_0b_2b_3b_4b_5b_7 \oplus b_0b_1b_2b_3b_6 \oplus b_0b_1b_2b_3b_7 \oplus b_0b_1b_2b_6b_7 \oplus b_0b_2b_3b_4b_6 \oplus b_1b_2b_3b_5b_7 \oplus b_0b_1b_3b_6b_7 \oplus b_0b_1b_3b_5b_7 \oplus b_0b_3b_5b_6b_7 \oplus b_1b_2b_5b_6b_7 \oplus b_1b_4b_5b_6b_7 \oplus b_0b_1b_2b_5b_7 \oplus b_1b_2b_4b_5b_6 \oplus b_0b_2b_3b_4b_5 \oplus b_0b_2b_3b_5b_6 \oplus b_0b_1b_5b_6b_7 \oplus b_1b_3b_4b_5b_6 \oplus b_2b_3b_4b_5b_7 \oplus b_1b_2b_3b_5b_6 \oplus b_1b_3b_5b_6b_7 \oplus b_0b_1b_2b_4b_5 \oplus b_0b_1b_2b_3b_4 \oplus b_1b_3b_4b_5b_7 \oplus b_0b_2b_4b_6b_7 \oplus b_0b_3b_4b_6b_7 \oplus b_2b_3b_4b_6b_7 \oplus b_1b_2b_4b_5b_7 \oplus b_0b_2b_3b_5b_7 \oplus b_2b_4b_5b_6b_7 \oplus b_0b_1b_4b_5b_6 \oplus b_0b_1b_3b_4b_6 \oplus b_2b_3b_4b_5b_6 \oplus b_1b_2b_3b_4b_6 \oplus b_3b_4b_5b_6b_7 \oplus b_0b_4b_5b_6b_7 \oplus b_0b_1b_6b_7 \oplus b_0b_3b_4b_5 \oplus b_3b_4b_6b_7 \oplus b_0b_2b_5b_7 \oplus b_0b_3b_5b_6 \oplus b_0b_3b_5b_7 \oplus b_0b_1b_3b_5 \oplus b_0b_2b_3b_4 \oplus b_0b_1b_3b_6 \oplus b_0b_1b_4b_6 \oplus b_1b_3b_4b_7 \oplus b_0b_1b_2b_6 \oplus b_2b_3b_5b_7 \oplus b_1b_2b_4b_6 \oplus b_2b_3b_4b_7 \oplus b_0b_3b_4b_6 \oplus b_1b_2b_3b_5 \oplus b_0b_3b_6b_7 \oplus b_0b_1b_4b_5 \oplus b_1b_2b_5b_7 \oplus b_0b_1b_5b_6 \oplus b_2b_3b_5b_6 \oplus b_0b_4b_5b_6 \oplus b_1b_3b_4b_5 \oplus b_0b_4b_6b_7 \oplus b_0b_1b_5b_7 \oplus b_0b_1b_3b_7 \oplus b_3b_5b_6b_7 \oplus b_3b_4b_5b_7 \oplus b_1b_2b_3b_4 \oplus b_1b_2b_6b_7 \oplus b_0b_3b_4b_7 \oplus$

$b_0b_3b_5 \oplus b_1b_4b_5 \oplus b_1b_2b_4 \oplus b_4b_6b_7 \oplus b_1b_3b_5 \oplus b_0b_3b_4 \oplus b_2b_3b_4 \oplus b_2b_3b_5 \oplus b_0b_2b_6 \oplus$
$b_3b_4b_5 \oplus b_3b_4b_7 \oplus b_1b_5b_7 \oplus b_2b_4b_7 \oplus b_5b_6b_7 \oplus b_3b_5b_6 \oplus b_3b_4b_6 \oplus b_3b_6b_7 \oplus b_4b_5b_7 \oplus$
$b_0b_2b_3 \oplus b_2b_4b_5 \oplus b_0b_4b_7 \oplus b_2b_5b_6 \oplus b_1b_6b_7 \oplus b_0b_4b_6 \oplus b_0b_4b_5 \oplus b_2b_3b_6 \oplus b_3b_5b_7 \oplus$
$b_0b_6b_7 \oplus b_2b_6b_7 \oplus b_4b_5 \oplus b_2b_5 \oplus b_3b_4 \oplus b_6b_7 \oplus b_1b_6 \oplus b_0b_6 \oplus b_0b_7 \oplus b_1b_4 \oplus b_3b_7 \oplus$
$b_0b_5 \oplus b_0b_4 \oplus b_3b_5 \oplus b_2b_3 \oplus b_5b_7 \oplus b_0b_1 \oplus b_1b_5 \oplus b_4b_6 \oplus b_2 \oplus b_3 \oplus b_0 \oplus b_5 \oplus b_1.$

$S_5(B_0) = b_0b_1b_2b_4b_5b_6b_7 \oplus b_0b_1b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_4b_5b_7 \oplus b_0b_1b_2b_3b_5b_7 \oplus b_1b_2b_4b_5b_6b_7 \oplus$
$b_0b_2b_3b_4b_6b_7 \oplus b_0b_1b_2b_3b_5b_6 \oplus b_0b_2b_4b_5b_6b_7 \oplus b_1b_2b_3b_4b_6b_7 \oplus b_0b_1b_2b_4b_5b_6 \oplus$
$b_0b_1b_2b_3b_4b_5 \oplus b_1b_2b_3b_5b_6b_7 \oplus b_0b_2b_4b_5b_7 \oplus b_0b_2b_3b_5b_7 \oplus b_0b_1b_5b_6b_7 \oplus b_1b_3b_4b_6b_7 \oplus$
$b_0b_2b_3b_6b_7 \oplus b_0b_1b_4b_5b_7 \oplus b_0b_1b_2b_4b_6 \oplus b_0b_3b_4b_5b_6 \oplus b_2b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_5 \oplus$
$b_0b_1b_2b_3b_7 \oplus b_1b_2b_4b_5b_6 \oplus b_0b_1b_4b_6b_7 \oplus b_0b_3b_4b_5b_7 \oplus b_0b_2b_3b_4b_7 \oplus b_0b_1b_2b_4b_5 \oplus$
$b_0b_4b_5b_6b_7 \oplus b_2b_3b_4b_5b_6 \oplus b_1b_3b_4b_5b_6 \oplus b_0b_1b_2b_3b_6 \oplus b_0b_1b_4b_5b_6 \oplus b_0b_1b_2b_3b_4 \oplus$
$b_1b_2b_3b_4b_6 \oplus b_2b_4b_5b_6b_7 \oplus b_0b_1b_2b_6b_7 \oplus b_0b_1b_3b_5b_7 \oplus b_1b_2b_3b_4b_7 \oplus b_3b_4b_5b_7 \oplus$
$b_0b_1b_3b_4 \oplus b_1b_4b_5b_7 \oplus b_1b_2b_4b_7 \oplus b_1b_2b_3b_5 \oplus b_0b_4b_6b_7 \oplus b_2b_3b_4b_7 \oplus b_0b_1b_4b_6 \oplus$
$b_0b_1b_5b_6 \oplus b_0b_1b_4b_5 \oplus b_0b_2b_4b_5 \oplus b_0b_2b_3b_4 \oplus b_3b_4b_6b_7 \oplus b_1b_3b_5b_6 \oplus b_1b_3b_4b_7 \oplus$
$b_0b_1b_4b_7 \oplus b_0b_1b_5b_7 \oplus b_0b_1b_2b_3 \oplus b_3b_4b_5b_6 \oplus b_0b_1b_2b_5 \oplus b_0b_1b_3b_7 \oplus b_0b_3b_4b_7 \oplus$
$b_2b_4b_5b_7 \oplus b_0b_2b_3b_5 \oplus b_1b_2b_4b_5 \oplus b_2b_3b_4b_6 \oplus b_1b_2b_3b_6 \oplus b_2b_3b_4b_5 \oplus b_1b_3b_4b_6 \oplus$
$b_0b_5b_6b_7 \oplus b_2b_3b_5b_6 \oplus b_1b_3b_5b_7 \oplus b_0b_4b_7 \oplus b_4b_5b_7 \oplus b_2b_4b_5 \oplus b_0b_5b_6 \oplus b_0b_1b_6 \oplus$
$b_5b_6b_7 \oplus b_0b_1b_3 \oplus b_0b_2b_6 \oplus b_1b_5b_7 \oplus b_2b_4b_7 \oplus b_2b_3b_7 \oplus b_1b_2b_5 \oplus b_1b_2b_4 \oplus b_0b_1b_2 \oplus$
$b_1b_3b_5 \oplus b_0b_3b_5 \oplus b_1b_4b_6 \oplus b_0b_2b_4 \oplus b_2b_3b_6 \oplus b_1b_3b_6 \oplus b_2b_5b_7 \oplus b_3b_5b_7 \oplus b_1b_3b_7 \oplus$
$b_2b_6b_7 \oplus b_0b_1b_5 \oplus b_1b_5b_6 \oplus b_1b_6b_7 \oplus b_1b_4b_7 \oplus b_0b_1b_7 \oplus b_0b_1b_4 \oplus b_3b_4b_5 \oplus b_1b_2b_6 \oplus$
$b_3b_4 \oplus b_1b_6 \oplus b_2b_4 \oplus b_1b_5 \oplus b_4b_6 \oplus b_5b_7 \oplus b_0b_3 \oplus b_7 \oplus b_6 \oplus \neg b_4.$

$S_6(B_0) = b_0b_1b_3b_4b_5b_6b_7 \oplus b_0b_1b_2b_4b_5b_6b_7 \oplus b_0b_1b_3b_4b_5b_7 \oplus b_0b_1b_2b_3b_5b_6 \oplus b_0b_1b_2b_3b_4b_7 \oplus$
$b_0b_1b_3b_4b_5b_6 \oplus b_0b_1b_2b_4b_6b_7 \oplus b_0b_1b_2b_4b_5b_7 \oplus b_0b_1b_3b_5b_6b_7 \oplus b_0b_1b_3b_4b_6b_7 \oplus$
$b_1b_3b_4b_5b_6b_7 \oplus b_0b_1b_2b_4b_5b_6 \oplus b_0b_1b_2b_5b_6b_7 \oplus b_0b_4b_5b_6b_7 \oplus b_1b_2b_3b_5b_7 \oplus b_1b_2b_3b_4b_5 \oplus$
$b_0b_1b_3b_6b_7 \oplus b_0b_1b_4b_5b_7 \oplus b_0b_1b_2b_3b_5 \oplus b_1b_4b_5b_6b_7 \oplus b_0b_3b_4b_6b_7 \oplus b_0b_2b_5b_6b_7 \oplus$
$b_0b_1b_2b_3b_7 \oplus b_1b_2b_4b_5b_6 \oplus b_0b_2b_4b_5b_7 \oplus b_2b_3b_4b_6b_7 \oplus b_0b_1b_2b_3b_6 \oplus b_1b_3b_4b_5b_6 \oplus$
$b_1b_2b_5b_6b_7 \oplus b_2b_3b_5b_6b_7 \oplus b_1b_2b_3b_4b_7 \oplus b_3b_4b_5b_6b_7 \oplus b_0b_2b_3b_5b_6 \oplus b_1b_3b_4b_5b_7 \oplus$
$b_0b_1b_3b_4b_5 \oplus b_0b_2b_4b_6b_7 \oplus b_0b_2b_3b_4b_5 \oplus b_0b_1b_3b_6 \oplus b_1b_2b_4b_7 \oplus b_1b_2b_3b_5 \oplus b_0b_3b_4b_7 \oplus$
$b_0b_1b_6b_7 \oplus b_4b_5b_6b_7 \oplus b_1b_3b_5b_7 \oplus b_0b_2b_4b_6 \oplus b_0b_2b_3b_5 \oplus b_1b_2b_3b_4 \oplus b_0b_3b_4b_6 \oplus$
$b_2b_3b_4b_5 \oplus b_2b_3b_5b_6 \oplus b_1b_2b_3b_6 \oplus b_1b_4b_6b_7 \oplus b_0b_1b_2b_5 \oplus b_0b_1b_3b_4 \oplus b_0b_2b_3b_7 \oplus$
$b_1b_4b_5b_7 \oplus b_1b_3b_4b_6 \oplus b_1b_3b_4b_7 \oplus b_0b_2b_4b_5 \oplus b_2b_3b_4b_6 \oplus b_1b_2b_4b_5 \oplus b_0b_2b_3b_6 \oplus b_0b_2b_5b_7 \oplus$
$b_2b_3b_5b_7 \oplus b_0b_1b_3b_7 \oplus b_0b_4b_6b_7 \oplus b_0b_1b_2b_4 \oplus b_3b_4b_6 \oplus b_0b_2b_5 \oplus b_0b_4b_7 \oplus b_1b_5b_6 \oplus$
$b_0b_1b_5 \oplus b_1b_2b_5 \oplus b_1b_3b_6 \oplus b_0b_3b_5 \oplus b_5b_6b_7 \oplus b_0b_1b_4 \oplus b_0b_4b_5 \oplus b_1b_6b_7 \oplus b_0b_1b_3 \oplus$
$b_0b_2b_4 \oplus b_0b_2b_6 \oplus b_3b_5b_7 \oplus b_2b_4b_6 \oplus b_0b_5b_6 \oplus b_4b_5b_6 \oplus b_0b_4b_6 \oplus b_2b_3b_7 \oplus b_1b_3b_4 \oplus$

$b_1b_4b_7 \oplus b_1b_2b_7 \oplus b_2b_4b_7 \oplus b_3b_6b_7 \oplus b_2b_3b_4 \oplus b_0b_5b_7 \oplus b_0b_3b_6 \oplus b_1b_4b_6 \oplus b_1b_2b_6 \oplus b_3b_5 \oplus b_0b_5 \oplus b_2b_3 \oplus b_5b_7 \oplus b_1b_7 \oplus b_1b_3 \oplus b_4b_6 \oplus b_3b_7 \oplus b_0b_4 \oplus b_0b_7 \oplus b_5 \oplus b_6 \oplus \neg b_3.$

$S_7(B_0) = b_0b_2b_3b_4b_5b_6b_7 \oplus b_0b_1b_3b_4b_5b_6b_7 \oplus b_0b_2b_4b_5b_6b_7 \oplus b_0b_2b_3b_4b_6b_7 \oplus b_0b_2b_3b_4b_5b_7 \oplus b_0b_1b_3b_4b_5b_7 \oplus b_0b_1b_2b_4b_5b_7 \oplus b_1b_3b_4b_5b_6b_7 \oplus b_0b_1b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_6b_7 \oplus b_0b_1b_3b_4b_6b_7 \oplus b_0b_2b_3b_5b_6b_7 \oplus b_0b_2b_3b_4b_5b_6 \oplus b_0b_1b_3b_4b_7 \oplus b_0b_3b_5b_6b_7 \oplus b_0b_1b_2b_3b_6 \oplus b_0b_2b_3b_4b_6 \oplus b_1b_2b_3b_5b_6 \oplus b_0b_1b_2b_3b_4 \oplus b_1b_4b_5b_6b_7 \oplus b_0b_1b_3b_4b_5 \oplus b_0b_1b_2b_5b_7 \oplus b_0b_2b_3b_4b_5 \oplus b_2b_3b_4b_5b_6 \oplus b_0b_1b_5b_6b_7 \oplus b_0b_1b_2b_4b_6 \oplus b_0b_2b_3b_4b_7 \oplus b_0b_1b_4b_5b_7 \oplus b_0b_3b_4b_6b_7 \oplus b_1b_2b_4b_5b_6 \oplus b_3b_4b_5b_6b_7 \oplus b_0b_1b_4b_5b_6 \oplus b_0b_3b_4b_5b_6 \oplus b_1b_2b_5b_6b_7 \oplus b_1b_2b_4b_6b_7 \oplus b_0b_1b_2b_6b_7 \oplus b_1b_3b_5b_6b_7 \oplus b_0b_1b_4b_7 \oplus b_0b_1b_2b_4 \oplus b_2b_4b_6b_7 \oplus b_1b_3b_4b_7 \oplus b_1b_4b_5b_7 \oplus b_4b_5b_6b_7 \oplus b_1b_2b_4b_6 \oplus b_2b_5b_6b_7 \oplus b_1b_5b_6b_7 \oplus b_1b_2b_3b_5 \oplus b_1b_2b_3b_7 \oplus b_0b_2b_4b_7 \oplus b_0b_2b_3b_5 \oplus b_0b_1b_2b_7 \oplus b_0b_3b_6b_7 \oplus b_0b_3b_4b_7 \oplus b_0b_1b_3b_6 \oplus b_0b_1b_2b_3 \oplus b_0b_3b_4b_6 \oplus b_3b_4b_5b_6 \oplus b_0b_3b_5b_6 \oplus b_1b_2b_3b_4 \oplus b_0b_2b_3b_7 \oplus b_0b_4b_6b_7 \oplus b_1b_4b_6b_7 \oplus b_0b_3b_5b_7 \oplus b_1b_2b_4b_5 \oplus b_0b_2b_3b_6 \oplus b_2b_3b_4b_7 \oplus b_1b_3b_6b_7 \oplus b_0b_1b_2b_5 \oplus b_0b_2b_4b_6 \oplus b_0b_1b_3b_4 \oplus b_1b_3b_6 \oplus b_2b_4b_6 \oplus b_1b_2b_6 \oplus b_0b_2b_7 \oplus b_1b_4b_7 \oplus b_0b_3b_6 \oplus b_0b_3b_7 \oplus b_2b_6b_7 \oplus b_1b_2b_3 \oplus b_0b_4b_5 \oplus b_2b_5b_6 \oplus b_0b_1b_5 \oplus b_0b_6b_7 \oplus b_0b_1b_6 \oplus b_4b_5b_7 \oplus b_0b_2b_3 \oplus b_0b_2b_5 \oplus b_3b_5b_7 \oplus b_0b_5b_6 \oplus b_0b_1b_4 \oplus b_0b_3b_5 \oplus b_3b_4b_5 \oplus b_4b_5b_6 \oplus b_2b_3b_5 \oplus b_1b_3b_5 \oplus b_4b_6b_7 \oplus b_0b_6 \oplus b_2b_4 \oplus b_2b_6 \oplus b_3b_5 \oplus b_1b_7 \oplus b_0b_2 \oplus b_5b_7 \oplus b_4b_6 \oplus b_1b_2 \oplus b_0b_7 \oplus b_7 \oplus b_4 \oplus b_5 \oplus b_2.$

Bibliography

[1] F.A. Aloul, A. Ramani, I.L. Markov, and K.A. Sakallah. Generic ilp versus specialized 0-1 ilp: An update. In *Proceedings of International conference on computer-aided design*, pages 450–457, 2002.

[2] T. Alsinet, a F. Many and J. Planes. An efficient solver for weighted Max-SAT. *Journal of Global Optimization*, 41(1):61–73, 2008.

[3] M. F. Anjos. Semidefinite optimization approaches for satisfiability and maximum-satisfiability problems. *Journal on Satisfiability, Boolean Modeling and Computation*, 1(1):1–47, 2006.

[4] C. Ansotegui, M. Bonet, and J. Levy. Solving Weighted partial MaxSAT through satisfiability testing. In *Proceedings of 12th International Conference on Theory and Applications of Satisfiability Testing*, 2009.

[5] C. Ansótegui, M. L. Bonet, and J. Levy. Solving (weighted) partial MaxSAT through satisfiability testing. In *Proceedings of International conference on Theory and Applications of Satisfiability Testing*, pages 427–440, 2009.

[6] J. Argelich, A. Cabiscol, I. Lynce, and F. Manyà. Modelling max-csp as partial Max-SAT. In *Proceedings of 11th International Conference on Theory and Applications of Satisfiability Testing*, pages 1–14, 2008.

[7] G. Audemard, P. Bertoli, A. Cimatti, A. Kornilowicz, and R. Sebastiani. A SAT based approach for solving formulas over boolean and linear mathematical propositions. In *Proceedings of International Conference on Automated Deduction*, pages 195–210, 2002.

[8] Bikramjit B. and Landon K. Coalition structure generation in multi-agent systems with mixed externalities. In *Proceedings of 9th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2010.

[9] N. Bansal and V. Raman. Upper bounds for MaxSat: Further improved. In *Proceedings of International symposium on algorithms and computation*, pages 247–258, 1999.

[10] D.L. Berre and A. Parrain. The SAT4J library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[11] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207, 1999.

[12] A. Biere, M. Heulu, H. Maaren, and T. Walsh. Handbook of satisfiability, 2009.

[13] B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2:299?–306, 1999.

[14] E. Catilina and R. Feinberg. Market power and incentives to form research consortia. *Review of Industrial Organization*, 28:129–144, 2008.

[15] Y. Chen, S. Safarpour, A. Veneris, and J. Marques-Silva. Spatial and temporal design debug using partial MaxSAT. In *Proceedings of ACM Great Lakes Symposium on VLSI*, pages 345–350, 2009.

[16] E. M. Clarke, D. Kroening, and F. Lerda. A tool for checking ansi-c programs. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems*, pages 168–176, 2004.

[17] V. Conitzer and T. Sandholm. Computing Shapley values, manipulating value division schemes, and checking core membership in multi-issue domains. In *Proceedings of 19th National Conference on Artificial Intelligence*, pages 219–225, 2004.

[18] V. Conitzer and T. Sandholm. Complexity of constructing solutions in the core based on synergies among coalitions. *Artificial Intelligence*, 170(6-7):607–619, 2006.

[19] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[20] M.C. Cooper, S. Cussat-Blanc, M. deRoquemaurel, and P. Régnier. Soft arc consistency applied to optimal planning. In *Proceedings of International conference on principles and practice of constraint programming*, pages 680–684, 2006.

[21] N. T. Courtois and G. V. Bard. Algebraic cryptanalysis of the data encryption standard. In *Proceedings of IMA International Conference on Cryptography and Coding*, pages 152–169, 2007.

[22] N. T. Courtois, G. V. Bard, and D. Wagner. Algebraic and slide attacks on KeeLoq. In *Proceedings of International Workshop of Fast Software Encryption*, pages 97–115, 2008.

[23] V. D. Dang, R. K. Dash, A. Rogers, and N. R. Jennings. Overlapping coalition formation for efficient data fusion in multi sensor network. In *Proceedings of 21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference*, 2006.

[24] J. Davies, J. Cho, and F. Bacchus. Using learnt clauses in MaxSAT. In *Proceedings of International conference on principles and practice of constraint programming*, pages 176–190, 2010.

[25] J. R. Douceur. The sybil attack. In *Proceedings of 1st International Workshop on Peer-to-Peer Systems*, pages 251–260, 2002.

[26] V. I. Dylkeyt, R. T. Faizullin, and I. G. Khnykin. Reducing the problem of asymmetric ciphers cryptanalysis to solving satisfiability problems. In *Proceedings of XIII All-Russian Conference Mathematical Methods in Pattern Recognition*, pages 249–251, 2007.

[27] T. Eibach, E.Pliz, and G. Volkel. Attacking Bivium using SAT solvers. In *Proceedings of International Symposium on the Theory and Applications of Satisfiability and Testing*, pages 63–76, 2008.

[28] MaxSAT Evaluations. http://maxsat.ia.udl.cat:81/.

[29] R. T. Faizullin, I. G. Khnykin, and V. I. Dylkeyt. The SAT solving method as applied to cryptographic analysis of asymmetric ciphers. *The Computing Research Repository*, abs/0907.1755, 2009.

[30] M. Felegyhazi and J. P. Hubaux. Game theory in wireless networks: a tutorial. In *EPFL Laboratory for Computer Communications and Applications*, pages 1–14, 2006.

[31] Federal Information Processing Standards Publication FIPS 197. Announcing the advanced encryption standard (AES), 2001.

[32] F. D. Garcia, G. K. Gans, R. Muijrers, P. Rossum, R. Verdult, R. W. Schreur, and B. Jacobs. Dismantling mifare classic. In *Proceedings of European Symposium on Research in Computer Security*, pages 97–114, 2008.

[33] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[34] J. A. Halderman, S. D. Schoen, N. A. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold-boot attacks on encryption keys. In *Proceedings of USENIX Security Symposium*, pages 45–60, California, USA, Jul 2008.

[35] P. Halmos and S. Givant. Introduction to boolean algebras, 2009.

[36] K. Haribabu, A. Paul, and C. Hota. Detecting sybils in peer-to-peer overlays using psychometric analysis methods. In *Workshops of International Conference on Advanced Information Networking and Applications*, pages 114 – 119, 2011.

[37] N. A. Heninger. *Error Correction and the Cryptographic Key*. PhD thesis, Univ. of Princeton, May 2011. ftp://ftp.cs.princeton.edu/reports/2011/897.pdf.

[38] F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: a new weighted Max-SAT solver. In *Proceedings of International Conference on Theory and Applications of Satisfiability Testing*, pages 41–55, 2007.

[39] F. Heras, J. Larrosa, and A. Oliveras. MiniMaxSat: An efficient weighted Max-SAT solver. *Journal of Articial Intelligence Research*, 31:1–32, 2008.

[40] E. Homsirikamol, P. Morawiecki, M. Rogawski, and M. Srebrny. Security margin evaluation of SHA-3 contest analists through SAT-based attacks. *Computer Information Systems and Industrial Management*, 7564:56–67, 2012.

[41] K. Honjyo and T. Tanjo. ShinMaxSat: A weighted partial Max-SAT solver inspired by MiniSat+. http://www.edu.kobe-u.ac.jp/istc-tamlab/cspsat/sms/.

[42] J. Håstad. Some optimal inapproximability results. In *Proceedings of 28th ACM Symposium on the Theory of Computing*, pages 1–10, 1997.

[43] S. Ieong and Y. Shoham. Marginal contribution nets: A compact representation scheme for coalitional games. In *Proceedings of 6th ACM Conference on Electronic Commerce*, 2005.

[44] D. Jackson, I. Schechter, and I. Shlyakhter. Alcoa: the alloy constraint analyzer. In *Proceedings of International Conference on Software Engineering*, pages 730–733, 2000.

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

[46] M. Jose and R. Majumdar. Bug-assist: Assisting fault localization in ANSI-C programs. In *Proceedings of International conference on computer aided verif ication*, pages 504–509, 2011.

[47] M. Jose and R. Majumdar. Cause clue clauses: Error localization using maximum satisfiability. In *Proceedings of ACM SIGPLAN conference on programming language design and implementation*, pages 437–446, 2011.

[48] F. Juma, E.I. Hsu, and S.A. McIlraith. Preference-based planning via MaxSAT. In *Proceedings of Canadian conference on AI*, pages 109–120, 2012.

[49] A.A. Kamal. Applications of SAT solvers to AES key recovery from decayed key schedule images. In *Proceedings of 4th International Conference on Emerging Security Information Systems and Technologies*, pages 216–220, 2010.

[50] H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT? In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science*, pages 406–415, 1997.

[51] M. Koshimura, H. Nabeshima, H. Fujita, and R. Hasegawa. Solving open job-shop scheduling problems by SAT encoding. *IEICE Transactions on Information and Systems*, 93-D(8):2316–2318, 2010.

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

[53] A. Kugel. Improved exact solver for the weighted Max-SAT problem. In *Proceedings of Pragmatics of SAT*, 2010.

[54] J. Larrosa, F. Heras, and S. de Givry. A logical approach to efficient Max-SAT solving. *Artificial Intelligence*, 172(2-3):204–233, 2008.

[55] J. Larrosa and T. Schiex. Solving weighted csp by maintaining arc consistency. *Artificial Inteligence*, 159(1-2):1–26, 2004.

[56] R. Letz. Lemma and model caching in decision procedures for quantified boolean formulas. In *Proceedings of International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 160–175, 2002.

[57] B. N. Levine, C. Shields, and N. B. Margolin. A survey of solutions to the sybil attack, 2006.

[58] C. M. Li, F. Manyà, and J. Planes. New inference rules for MaxSAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.

[59] C.M. Li, F. Manya, N.O. Mohamedou, and J. Planes. Exploiting cycle structures in Max-Sat. In *Proceedings of 12th International Conference on Theory and Applications of Satisfiability Testing*, 2009.

[60] X. Liao, D. Hao, and K. Sakurai. Achieving cooperative detection against sybil attack in wireless ad hoc networks: A game theoretic approach. In *Proceedings of 17th Asia-Pacific Conference on Communications*, pages 806 – 811, 2011.

[61] X. Liao, D. Hao, and K. Sakurai. Classification on attacks in wireless ad hoc networks: A game theoretic view. In *Proceedings of 7th International Conference on Networked Computing and Advanced Information Management*, pages 144–149, 2011.

[62] H. Lin, K. Su, and C. M. Li. Within-problem learning for efficient lower bound computation in Max-SAT solving. In *Proceedings of 22th AAAI Conference on Artificial Intelligence*, pages 351–356, 2008.

[63] H. Mangassarian, A.G. Veneris, S. Safarpour, F.N. Najm, and M.S. Abadir. Maximum circuit activity estimation using pseudo-boolean satisfiability. In *Proceedings of Conference on design, automation and test in Europe*, pages 1538–1543, 2007.

[64] V. Manquinho, J. Marques-Silva, and J. Planes. Algorithms for weighted boolean optimization. In *Proceedings of International conference on Theory and Applications of Satisfiability Testing*, pages 495–508, 2009.

[65] M. H. Manshaei, Q. Zhu, T. Alpcan, T. Basar, and J. P. Hubaux. Game theory meets network security and privacy. *ACM Computing Surveys*, 45(25):1–39, 2013.

[66] J. Marques-Silva. Practical applications of boolean satisfiability. In *Proceedings of 9th International Workshop on Discrete Event Systems*, pages 28–30, 2008.

[67] R. Martins, V. Manquinho, and I. Lynce. Parallel search for maximum satisfiability. *AI Communications*, 25:75–95, 2012.

[68] T. Michalak, A. Dowell, P. McBurney, and M. Wooldridge. Optimal coalition structure generation in partition function games. In *Proceedings of European Conference on Artificial Intelligence*, 2008.

[69] T. Michalak, D. Marciniak, M. Szamotulski, T. Rahwan, M. Wooldridge, P. McBurney, and N. Jennings. A logic-based representation for coalitional games with externalities. In *Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems*, 2010.

[70] I. Mironov and L. Zhang. Applications of SAT solvers to cryptanalysis of hash functions. In *Proceedings of International Symposium on the Theory and Applications of Satisfiability and Testing*, pages 102–115, 2006.

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

[72] R. Muhammad and P. J. Stuckey. A stochastic non-CNF SAT solver. In *Proceedings of Trends in Artificial Intelligence, 9th Pacific Rim International Conference on Artificial Intelligence*, pages 120–129, 2006.

[73] J. Newsome, E. Shi, D. Song, and A. Perrig. The sybil attack in sensor networks: analysis & defenses. In *Proceedings of 3rd International Symposium on Information Processing in Sensor Networks*, pages 259–268, 2004.

[74] R. Nieuwenhuis and A. Oliveras. On SAT modulo theories and optimization problems. In *Proceedings of International conference on theory and applications of satisf iability testing*, pages 156–169, 2006.

[75] N. Ohta, V. Conitzer, R. Ichimura, Y. Sakurai, A. Iwasaki, and M. Yokoo. Coalition structure generation utilizing compact characteristic function representation. In *Proceedings of 15th International Conference on Principles and Practice of Constraint Programming (CP '09)*, 2009.

[76] M. J. Osborne and A. Rubinstein. *A Course in Game Theory.* The MIT Press, 1994.

[77] C. H. Papadimitriou. Computational complexity, 1994.

[78] J.D. Park. Using weighted MAX-SAT engines to solve mpe. In *Proceedings of 16th AAAI Conference on Artificial Intelligence*, pages 682–687, 2002.

[79] C. Patsakis. RSA private key reconstruction from random bits using SAT solvers. *IACR Cryptology ePrint Archive*, 26, 2013.

[80] C. Patsakis. RSA private key reconstruction from random bits using SAT solvers. *IACR Cryptology ePrint Archive*, 2013.

[81] D. N. Pham, J. R. Thornton, and A. Sattar. Building structure into local search for SAT. In *Proceedings of 20th International Joint Conference on Artificial Intelligencef*, pages 2359–2364, 2007.

[82] K. Pipatsrisawat and A. Darwiche. Clone: Solving weighted Max-SAT in a reduced search space. In *Proceedings of 20th Australian Joint Conference on Artificial Intelligence*, pages 223–233, 2007.

[83] J. Planes. Improved branch and bound algorithms for Max-2-SAT and weighted Max-2-SAT. In *Proceedings of 9th International Conference on Principles and Practice of Constraint Programming*, page 991, 2003.

[84] J. Plasmans, J. Engwerda, B. Aarle, B. Bartolomeo, and T. Michalak. *Dynamic Modeling of Monetary and Fiscal Cooperation Among Nations*. Springer, 2006.

[85] T. Rahwan and N.R. Jennings. Coalition structure generation: Dynamic programming meets anytime optimisation. In *Proceedings of the 22th AAAI Conference on Artificial Intelligence*, 2008.

[86] T. Rahwan and N.R. Jennings. An improved dynamic programming algorithm for coalition structure generation. In *Proceedings of 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2008.

[87] T. Rahwan, T. Michalak, M. Wooldridge, and N. R. Jennings. Anytime coalition structure generation in multi-agent systems with positive or negative externalities. *Artificial Intelligence*, 186:95–122, 2012.

[88] T. Rahwan, T. P. Michalak, E. Elkind, P. Faliszewski, J. Sroka, M. Wooldridge, and N.R. Jennings. Constrained coalition formation. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, 2011.

[89] T. Rahwan, T.P. Michalak, and N.R. Jennings. A hybrid algorithm for coalition structure generation. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2012.

[90] T. Rahwan, S.D. Ramchurn, N.R. Jennings, and A. Giovannucci. An anytime algorithm for optimal coalition structure generation. *Journal of Artificial Intelligence Research*, 34:521–567, 2009.

[91] J. Rintanen, K. Heljanko, and I. Niemelä. Planning as satisfiability: parallel plans and algorithms for plan search. *Artificial Intelligence*, 170(12-13):1031–1080, 2006.

[92] N. Robinson, C. Gretton, D.N. Pham, and A. Sattar. Partial weighted MaxSAT for optimal planning. In *Proceedings of Pacif ic rim international conference on artificial intelligence*, pages 231–243, 2010.

[93] M.H. Rothkopf, A. Pekec, and R.M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44:1131–1147, 1998.

[94] S. Roy, C. Ellis, S. G. Shiva, D. Dasgupta, V. Shandilya, and Q. Wu. A survey of game theory as applied to network security. In *Proceedings of 43rd Hawaii International Conference on System Sciences*, pages 1–10, 2010.

[95] S. Safarpour, H. Mangassarian, A. Veneris, M.H. Liffiton, and K.A. Sakallah. Improved design debugging using maximum satisfiability. In *Proceedings of 7th Conference on Formal methods in computer-aided design*, pages 13–19, 2007.

[96] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proceedings of International joint conference on artificial intelligence*, pages 542–547, 1999.

[97] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme. Coalition structure generation with worst case guarantees. *Artificial Intelligence*, 111:209–238, 1999.

[98] T. Sandholm and V.R. Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94:99–137, 1997.

[99] B. Selman and H. Kautz. Planning as satisfiability. In *Proceedings of European Conference on Artificial Intelligence*, pages 359–363, 1992.

[100] S. Sen and P.S. Dutta. Searching for optimal coalition structures. In *Proceedings of 4th International Conference on Multi-Agent Systems*, 2000.

[101] T.C. Service and J.A. Adams. Constant factor approximation algorithms for coalition structure generation. *Autonomous Agents and Multi-Agent Systems*, 23:1–17, 2011.

[102] M. Sheeran, S. Singh, and G. Stalmarck. Checking safety properties using induction and a SAT solver. In *Proceedings of 3rd International Conference on Formal Methods in Computer-Aided Design*, pages 108–125, 2000.

[103] O. Shehory and S. Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101:165–200, 1987.

[104] M. Soos, K. Nohl, and C. Castelluccia. Extending SAT solvers to cryptographic problems. In *Proceedings of International Symposium on the Theory and Applications of Satisfiability and Testing*, pages 244–257, 2009.

[105] Z. Stachniak. Going non-clausal. In *Proceedings of 5th International Symposium on Theory and Applications of Satisfiability Testing*, pages 316–322, 2002.

[106] C. Thiffault, F. Bacchus, and T. Walsh. Solving non-clausal formulas with DPLL search. In *Proceedings of 7th International Conference on Theory and Applications of Satisfiability Testing*, pages 147–156, 2004.

[107] C. Thiffault, F. Bacchus, and T. Walsh. Solving non-clausal formulas with DPLL search. In *Proceedings of 7th International Conference on Theory and Applications of Satisfiability Testing*, pages 147–156, 2004.

[108] G. S. Tseitin. On the complexity of derivation in propositional calculus. *Automation of Reasoning: Classical Papers in Computational Logic*, 2:466–483, 1983.

[109] A. Tsow. An improved recovery algorithm for decayed AES key schedule images. In *Proceedings of Selected Areas in Cryptography*, pages 215–230, 2009.

[110] M. Tsvetovat and K. Sycara. Customer coalitions in the electronic marketplace. In *Proceedings of the fourth international conference on Autonomous agents*, 2000.

[111] S. Ueda, T. Hasegawa, N. Hashimoto, N. Ohta, A. Iwasaki, and M. Yokoo. Handling negative value rules in MC-net-based coalition structure generation. In *Proceedings of 11th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2012.

[112] S. Ueda, M. Kitaki, A. Iwasaki, and M. Yokoo. Concise characteristic function representations in coalitional games based on agent types. In *Proceedings of 22nd International Joint Conference on Artificial Intelligence*, 2011.

[113] M. Vasquez and J. Hao. A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Journal of Computational Optimization and Applications*, 20(2):137–157, 2001.

[114] Z. Xing and W. Zhang. Maxsolver: an efficient exact algorithm for (weighted) maximum satisfiability. *Artificial Intelligence*, 164(1-2):47–80, 2005.

[115] H Xu, R. Rutenbar, and K. Sakallah. Sub-SAT: A formulation for relaxed boolean satisfiability with applications in routing. In *Proceedings of International symposium on physical design*, pages 182–187, 2002.

[116] L. Zhang and F. Bacchus. MaxSAT heuristics for cost optimal planning. In *Proceedings of 26th AAAI Conference on Artificial Intelligence*, pages 1846–1852, 2012.

[117] X. Zhang and K. K. Parhi. High-speed vlsi architectures for the AES algorithm. *IEEE Transactions on Very Large Scale Integration VLSI Systems*, 12(9):957–967, 2004.

List of Related Publications

1. X. Liao, M. Koshimura, H. Fujita and R. Hasegawa, Extended MaxSAT to Solve the Coalition Structure Generation Problem with Externalities based on Agent Relations, *IEICE Transactions on Information and Systems*, conditionally accepted, 2013.

2. X. Liao, M. Koshimura, H. Fujita and R. Hasegawa, MaxSAT Encoding for MC-net-based Coalition Structure Generation Problem with Externalities, *IEICE Transactions on Information and Systems*, conditionally accepted, 2013.

3. X. Liao, H. Zhang, M. Koshimura, H. Fujita and R. Hasegawa, Using MaxSAT to Correct Errors in AES Key Schedule Images, *IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 284–291, 2013.

4. X. Liao, M. Koshimura, H. Fujita and R. Hasegawa, Solving the Coalition Structure Generation Problem MaxSAT, *IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 910–915, 2012.

5. X. Liao, D. Hao and K. Sakurai, Using Game Theory to Classify Wireless Ad Hoc Network Attacks with Analysis on Countermeasures, *International Journal of Advancements in Computing Technology (IJACT)*, 3(8): 296–303, 2011.

6. X. Liao, D. Hao and K. Sakurai, Achieving Cooperative Detection against Sybil Attack in Wireless Ad hoc Networks: A Game Theoretic Approach, *17th Asia-Pacific Conference on Communications (APCC)*, pages 806–811, 2011.

7. X. Liao, D. Hao and K. Sakurai, Classification on Attacks in Wireless Ad Hoc Networks: A Game Theoretic View, *7th International Conference on Networked Computing and Advanced Information Management (NCM)*, pages 144–149, 2011.

8. D. Hao, X. Liao, A. Adhikari, K. Sakurai and M. Yokoo: A Repeated Game Approach for Analyzing the Collusion on Selective Forwarding in Multihop Wireless Networks, *Computer Communications (ComCom)*, 35(17): 2125–2137, 2012.