Stochastic Boolean Satisfiability: Decision Procedure, Generalization, and Applications

Student: Nian-Ze Lee Advisor: Dr. Jie-Hong Roland Jiang

Graduate Institute of Electronics Engineering

National Taiwan University

Abstract

Stochastic Boolean satisfiability (SSAT) is an expressive language to formulate computational problems with randomness, such as the inference of Bayesian networks, propositional probabilistic planning, and partially observable Markov decision process (POMDP). Solving an SSAT formula lies in the PSPACE-complete complexity class, the same as solving a *quantified Boolean formula* (QBF). Despite its broad applications and profound theoretical values, SSAT has drawn relatively less attention compared to SAT or QBF.

We identify new applications of SSAT in the formal verification of *probabilistic design*. Probabilistic design, as well as approximate design, is an emerging computational paradigm to accept the device variability of VLSI systems in the nanometer regime, which imposes serious challenges to the design and manufacturing of reliable

DOI

systems. Despite recent intensive study on approximate design, probabilistic design receives relatively few attentions. We formulate the framework of *probabilistic property evaluation* for probabilistic design and exploit random-exist and exist-random quantified SSAT solving to approach the average-case and worst-case analyses, respectively. To the best of our knowledge, this is the first attempt that applies SSAT to analyze VLSI systems.

Motivated by the above emerging applications, we propose novel algorithms to solve random-exist and exist-random quantified SSAT formulas. These two fragments of SSAT have important AI applications in the reasoning of Bayesian networks and the search of an optimal strategy for probabilistic conformant planning. In contrast to the conventional SSAT-solving approaches based on Davis-Putnam-Logemann-Loveland (DPLL) search, we take advantage of modern techniques for SAT/QBF solving and model counting to improve the computational efficiency. Moreover, unlike the prior DPLL-based exact algorithms for SSAT solving, the proposed algorithms are able to solve an SSAT formula approximately by deriving upper or lower bounds of its satisfying probability.

The proposed algorithms are implemented in the open-source SSAT solvers `reSSAT` and `erSSAT`. Our evaluation shows that `reSSAT` and `erSSAT` solvers outperform the state-of-the-art SSAT solver on various formulas in both run-time and memory usage. They also provide useful bounds for cases where the state-of-the-art solver fails to compute exact answers.

In spite of its various applications, SSAT is limited by its descriptive power within

DOI

the PSPACE complexity class. More complex problems, such as the NEXPTIME-complete decentralized POMDP (Dec-POMDP), cannot be succinctly encoded with SSAT. To provide a logical formalism for such problems, we extend the *dependency quantified Boolean formula* (DQBF), a representative problem in the NEXPTIME-complete class, to its stochastic variant, named *dependency SSAT* (DSSAT), and show that DSSAT is also NEXPTIME-complete. We demonstrate potential applications of DSSAT in the synthesis of probabilistic/approximate design and the encoding of Dec-POMDP problems. We hope to encourage solver development with the established theoretical foundations.

Keywords: Stochastic Boolean satisfiability (SSAT); Model counting; Dependency quantified Boolean formula (DQBF); NEXPTIME-completeness; Probabilistic/Approximate design; Decentralized partially observable Markov decision process (Dec-POMDP)

DOI

論隨機布林可滿足性：

決策程序、廣義化、與應用

研究生: 李念澤　　指導教授: 江介宏　博士

國立臺灣大學電子工程學研究所

摘要

隨機布林可滿足性（SSAT）是一種豐富的邏輯語言，可用於表達具有機率性的計算問題，例如貝氏網絡推論，命題式機率性規劃，和部分可觀察馬可夫決策過程（POMDP）。求解SSAT公式之複雜度屬於多項式空間完全性複雜度類別（PSPACE-complete complexity class），與求解量化布林公式（QBF）相同。儘管具有廣泛的應用和深刻的理論價值，SSAT在文獻中所引起的關注較SAT或QBF稀少。

我們在機率性設計的正規驗證問題中找到SSAT的新應用。機率性設計以及近似性設計是一種新興的計算準則，可用來容忍VLSI系統在奈米製程下的元件變異性。儘管相關文獻對近似性設計進行了許多探討，機率性設計則很少受到關注。我們爲機率性設計提出了機率性質評估框架並利用隨機存在和存在隨機的量化SSAT公式求解，分別進行平均情況和最壞情況分析。據我們所知，這是文獻中首次將SSAT應用於VLSI系統分析。

受以上VLSI系統應用的推動，我們提出了新的演算法來求解隨機存在和存在隨機的量化SSAT公式。不同於與基於Davis-Putnam-Logemann-

DOI

Loveland（DPLL）搜尋的傳統SSAT演算法， 我們利用當代SAT或QBF求解技術和模型計數以提高計算效率。 與之前基於DPLL搜尋的準確SSAT演算法不同， 我們的演算法能夠近似地求解SSAT公式，計算其滿足機率的上下限。

我們在開源SSAT求解器reSSAT和erSSAT中實作所提出的新演算法。 實驗結果顯示reSSAT和erSSAT求解器的性能優於文獻中的SSAT求解器。 在文獻中的求解器無法計算出準確答案的情況下，它們也能提供有用的上下限。

儘管在不同領域都有許多應用，SSAT仍受限於其PSPACE complexity class的描述能力。 更複雜的問題，例如非確定性指數時間完全性（NEXPTIME-complete）的分散式POMDP（decentralized POMDP, Dec-POMDP），則無法用SSAT簡潔地表示。 爲了提供此類問題一個邏輯框架，我們借鏡了同爲NEXPTIME-complete的依賴性QBF（dependency QBF, DQBF）。 我們推廣SSAT並命名該推廣框架爲依賴性SSAT（dependency SSAT, DSSAT），同時證明DSSAT也是NEXPTIME-complete。 我們展示了DSSAT在機率或近似性設計的合成中的潛在應用，以及針對Dec-POMDP問題的編碼。 我們希望通過建立理論基礎來鼓勵DSSAT求解器的開發。

關鍵字： 隨機布林可滿足性問題、 模型計數、 依賴性量化布林公式、 非確定指數時間完全性、 機率性或近似性設計、 分散式部份可觀察馬可夫決策過程

DOI

Acknowledgements

Thank you all.

(This part will be finished before submitting the dissertation to NTU library.)

李念澤

Nian-Ze Lee

National Taiwan University

June 2021

DOI

Contents

Abstract **i**

Chinese Abstract **iv**

Acknowledgements **vi**

1 Introduction **1**

1.1 Motivation and the research needs 1

1.2 Our contributions 5

1.3 An overview of the dissertation 8

2 Related Work **10**

2.1 Probabilistic/Approximate design 10

2.2 Stochastic Boolean satisfiability 13

2.3 Model counting 14

3 Background **16**

3.1 Propositional logic 16

DOI

3.1.1 Conjunctive and disjunctive normal form 17
3.1.2 Boolean satisfiability 18
3.2 Model counting 20
3.2.1 Exact/Approximate model counting 20
3.2.2 Weighted model counting 20
3.3 Stochastic Boolean satisfiability 21

4 Probabilistic Design Evaluation 24

4.1 Preliminaries 25
4.1.1 Boolean network 25
4.1.2 Probability and random variables 26
4.2 Modeling probabilistic design 27
4.2.1 Probabilistic Boolean network 28
4.2.2 Probabilistic property evaluation 29
4.2.3 Extension to sequential probabilistic design 32
4.3 Solving probabilistic property evaluation 33
4.3.1 Solving MPPE and PPE via SSAT 33
4.3.2 Solving PPE via weighted model counting 40
4.3.3 Solving PPE via probabilistic model checking 45
4.4 Discussion 47
4.4.1 Probabilistic equivalence checking 47
4.4.2 Prioritized output requirement 48
4.4.3 Connection to approximate design analysis 49

DOI

4.5 Evaluation 49
4.5.1 Benchmark set 50
4.5.2 Experimental setup 53
4.5.3 Results 54

5 Random-Exist Quantified SSAT 63

5.1 Preliminaries 63
5.1.1 Generalization of SAT/UNSAT minterms 64
5.2 Solving random-exist quantified SSAT 65
5.2.1 Minimal satisfying assignment 68
5.2.2 Minimal conflicting assignment 69
5.2.3 Weight computation 69
5.2.4 Modification for approximate SSAT 70
5.3 Applications 73
5.3.1 Probability of success in planning 73
5.3.2 Probabilistic circuit verification 74
5.4 Evaluation 74
5.4.1 Benchmark set 75
5.4.2 Experimental setup 77
5.4.3 Results 78

6 Exist-Random Quantified SSAT 86

6.1 Preliminaries 86

DOI

6.1.1 Solving E-MAJSAT with weighted model counting 87
6.1.2 Clause selection 87
6.2 Clause-containment learning for E-MAJSAT 89
6.2.1 Enhancement techniques 94
6.2.2 Implementation details 102
6.3 Evaluation 103
6.3.1 Benchmark set 104
6.3.2 Experimental setup 106
6.3.3 Results 106

7 Dependency SSAT 119

7.1 Preliminaries 119
7.1.1 Dependency quantified Boolean formula 119
7.1.2 Decentralized POMDP 121
7.2 Lifting SSAT to NEXPTIME-completeness 123
7.2.1 Formulation 123
7.2.2 Complexity proof 125
7.3 Applications of DSSAT 127
7.3.1 Analyzing probabilistic/approximate partial design 127
7.3.2 Modeling Dec-POMDP 131

8 Conclusion and Future Work 141

List of Figures 144

DOI

List of Tables 146

List of Algorithms 148

Bibliography 149

DOI

Chapter 1

Introduction

1.1 Motivation and the research needs

Boolean satisfiability (SAT) [10] has been successfully applied to numerous research fields including artificial intelligence [83, 93], electronic design automation [71, 108], software verification [7, 45], etc. The tremendous benefits have encouraged the development of more advanced decision procedures for satisfiability with respect to more complex logics beyond pure propositional. For example, solvers of the satisfiability modulo theories (SMT) [6, 29] accommodate first order logic fragments; quantified Boolean formula (QBF) [13, 82] allows both existential and universal quantifiers; stochastic Boolean satisfiability (SSAT) [63, 66] models uncertainty with random quantification; and dependency QBF (DQBF) [5, 98] equips Henkin quantifiers to

DOI

describe multi-player games with partial information. Due to their simplicity and generality, various satisfiability formulations are under active investigation.

Among various generalizations of Boolean satisfiability, *stochastic Boolean satisfiability* (SSAT) [66] is a logical formalism for problems endowed with randomness. First formulated by Papadimitriou, SSAT is interpreted as *games against nature* [86]. Nondeterministic factors are introduced into the world of propositional logic through the creation of the *randomized quantifier*. A Boolean variable x can be randomly quantified with a probability $p \in [0, 1]$ in an SSAT formula by a randomized quantifier $\exists^p$ that requires x to take the Boolean value TRUE with probability p and FALSE with probability $1 - p$. Taking advantage of randomized quantifiers, a variety of computational problems inherent with uncertainty can be encoded into SSAT formulas, such as propositional probabilistic planning [62], Bayesian-network inference [3, 24, 44], and the analysis of partially observable Markov decision process [69].

While SSAT has been employed to solve various AI problems, to the best of our knowledge, it has not yet been applied to analyze VLSI systems, and how VLSI systems would benefit from the probabilistic reasoning of SSAT remains unclear. Conventionally, uncertain system behavior is undesirable and would be mitigated by employing techniques such as error detection [22] and error correction [77]. Nevertheless, in the post-Moore's era, the variability and uncertainty of manufacturing at the atomic level make devices under miniaturization sensitive to process variation and environmental fluctuation. As a result, the manufactured ICs may exhibit

DOI

uncertain probabilistic behavior, which imposes serious challenges to the design of reliable systems.

Recent research efforts have been made to accept the inevitable imperfection of devices based on the notions of *approximate design* and *probabilistic design*. In both notions, a system's behavior may deviate from its expected specification; however, this deviation is deterministic in the former case but probabilistic in the latter case. Despite the advancements made by prior endeavors, the analysis and synthesis of probabilistic design have gained relatively less attention. Hence, **there is a research need of a framework to evaluate probabilistic design**, and SSAT stands up as a suitable logical formalism to address the need. (In the following, the term "design" is used as a general term to refer to a single design instance, a set of design instances, or design process; the term "synthesis" is referred to as the design automation process transforming a system under design from high-level system specification to low-level circuit implementation.)

SSAT is closely related to two generalizations of Boolean satisfiability: *model counting* of propositional formulas and *quantified Boolean formulas* (QBFs). Given a propositional formula, model counting asks to compute the number of its satisfying assignments. In the weighted version, weights are assigned to the Boolean variables in the formula, and the goal is to compute the summation of weights of the satisfying assignments. Algorithms for model counting are under active development in recent years. In addition to *exact* model counting [95, 96], *approximate* model counting [16,

DOI

37, 38] has been investigated to improve scalability by relaxing exactness. On the other hand, from the perspective of the computational complexity, solving an SSAT formula lies in the PSPACE-complete [102] complexity class, the same as solving a QBF. Many endeavors have been invested in the algorithmic improvement [13] and solver evaluation [82] for QBF.

Nevertheless, in spite of its broad applications and profound theoretical values, SSAT has drawn relatively little attention compared to SAT, model counting, or QBF. Most prior efforts for SSAT solving are based on the conventional Davis-Putnam-Logemann-Loveland (DPLL) search [28], which suffers from the scalability issue when problem sizes grow. Therefore, **there is a research need to develop novel algorithms to enhance the scalability of SSAT solving**, and the recent advancements of SAT/QBF solving and model counting can be leveraged to help the algorithm design.

In spite of its rich expressiveness to encode problems ranging from AI to VLSI, SSAT is limited by its descriptive power within the PSPACE complexity class. More complex problems with nondeterminism might not be succinctly modeled as SSAT formulas. As a result, **there is a research need of a logical formalism for problems beyond PSPACE and with uncertainty.**

Finally, most research work regarding SSAT solving was done before the year 2010 [64, 66–69, 103]. Open-source implementations and SSAT instances for testing are barely available, which hinder the understanding of the algorithmic details and

DOI

empirical solver comparison. Consequently, **there is a need to provide open-source implementations and databases of SSAT instances to facilitate convenient evaluation of different algorithms and drive further advancements.**

1.2 Our contributions

This dissertation aims at contributing to the aforementioned research needs.

First, we approach the analysis of probabilistic design by formalizing the problem of *probabilistic property evaluation*. Different computational solutions are provided for the problem. Particularly, random-exist and exist-random quantified SSAT formulas are exploited to solve the average-case and worst-case analyses, respectively. To the best of our knowledge, this is the first attempt that applies SSAT to the analyze VLSI systems. (In the following, the terms "analysis" and "evaluation" are used interchangeably as general terms referring to the process of determining qualitative or quantitative properties of a design. We will formulate the problem of *probabilistic property evaluation*, and refer to the term "evaluation" as computing the satisfying probability of certain properties of a probabilistic design.)

Second, in contrast to the previous DPLL-based algorithms, we utilize modern techniques of SAT/QBF solving and model counting to improve SSAT solving. Motivated by the new VLSI applications, we focus on random-exist and exist-random

DOI

quantified fragments of SSAT formulas.

The random-exist quantified SSAT formula is of the form $\Phi = \text{ꓤ}X, \exists Y.\phi$, which is the counterpart of the forall-exist QBF. It has applications in Bayesian-network inference [3, 24]. We propose an algorithm that uses modern SAT solvers [31, 32] as plug-in engines. In addition to SAT solving, we also incorporate weighted model counting, which has been widely used in probabilistic inference [18, 97], to tackle randomized quantifiers. The randomized quantification in an SSAT formula can be approached with weighted model counting by assigning the weight of a variable quantified by ꓤ^p to be p. The proposed algorithm uses an SAT solver and a model counter in a *stand-alone* manner, leaving the internal structures of these solvers intact. Due to the stand-alone usage of these solvers, the proposed algorithm may directly benefit from the advancement of the solvers without any modification.

The exist-random quantified SSAT formulas has the form $\Phi = \exists X, \text{ꓤ}Y.\phi$, which is also known as *E-MAJSAT* [62]. Computational problems, such as computing a maximum-a-posteriori (MAP) hypothesis or a maximum expected utility (MEU) solution [30] in Bayesian networks, and searching an optimal plan for probabilistic conformant planning domains [62], can be formulated with E-MAJSAT. Inspired by the *clause-selection* [43, 90] technique, which is recently devised for QBF solving and becomes the state-of-the-art, we propose a learning method based on the *clause-containment principle* to solve E-MAJSAT. To the best of our knowledge, this is the first attempt to adopt QBF approaches for SSAT solving.

DOI

Moreover, the proposed algorithms solve an SSAT formula in a gradual manner that converges from approximate bounds of the satisfying probability to the exact answer. Therefore, they are able to provide useful information even if the exact answer is unavailable.

Third, to provide a logical formalism for more complex problems with uncertainty, we extend *dependency QBF* (DQBF) [5, 98] to the stochastic domain in view of the close relation between QBF and SSAT. DQBF is a representative problem in the NEXPTIME-complete [88] complexity class. It equips QBF with Henkin quantifiers to describe multi-player games with partial information. We formalize the problem of *dependency SSAT* (DSSAT) as a generalization for SSAT. We prove that DSSAT has the same NEXPTIME-complete complexity as DQBF, and therefore it can succinctly encode decision problems with uncertainty in the NEXPTIME complexity class. We demonstrate the potential applications of DSSAT to the synthesis of probabilistic and approximate design and the encoding of decentralized POMDP (Dec-POMDP) [84] problems. Our theoretical results would encourage the solver development.

Fourth, our implementation of the proposed SSAT algorithms is open-source, which helps other researchers to understand the details of the approaches and serves as a reference for the empirical evaluation of different algorithms.

DOI

1.3 An overview of the dissertation

The structure of this dissertation is outlined as follows.

- In Chapter 2, a brief survey of the literature is provided to highlight the advancements made in this dissertation.
- In Chapter 3, background knowledge required throughout the dissertation is discussed. Specific material for an individual chapter will be introduced when it is needed.
- In Chapter 4, a formal framework to evaluate properties of probabilistic design is proposed. Especially, random-exist and exist-random quantified SSAT formulas are exploited to solve the formulation. This chapter is based on our journal paper [54] published in IEEE Transactions on Computers.
- In Chapter 5, modern SAT-solving and model-counting techniques are combined to solve random-exist quantified SSAT formulas. This chapter is based on our conference paper [56] published at IJCAI '17.
- In Chapter 6, a clause-learning technique inspired by *clause selection*, a prevailing method recently invented for QBF, is devised to solve exist-random quantified SSAT formulas. This chapter is based on our conference paper [57] published at IJCAI '18.
- In Chapter 7, SSAT is lifted from the PSPACE-complete complexity class to

DOI

the NEXPTIME-completeness. We show the applicability of the lifted formalism to the analysis of probabilistic design and decentralized POMDP. This chapter is based on our conference paper [55] published at AAAI '21.

- In Chapter 8, we give concluding remarks and point out some potential directions for future investigation.

DOI

Chapter 2

Related Work

This chapter provides an overview of the literature related to this dissertation.

2.1 Probabilistic/Approximate design

While the shrinkage of device's feature size according to Moore's law [79] has driven the prosperity of microelectronic industry, the variability and uncertainty of devices at the atomic level pose serious challenges on the continuation of the law.

New computational paradigms are proposed in response to the challenges in the post-Moore's era. Among many other efforts, along the direction of *approximate design*, circuit architectures for reconfigurable adders whose accuracy can be adjusted based on application scenarios [47, 109] and energy-efficient adders with a moderate

DOI

error rate [48] are proposed. The automatic synthesis and analysis of approximate circuits have also been studied [59, 72, 73, 80, 91, 106, 107]. On the other hand, for *probabilistic design*, Chakrapani et al. [17] consider CMOS devices with probabilistic behavior, establish the relation between energy consumption and probability of correct switching, and exploit it to trade energy against correctness.

Despite the advancements made by prior endeavors, most of them focus on approximate design. The analysis and synthesis of probabilistic design have gained relatively less attention. Nevertheless, the study of probabilistic design is important in the following respects. First, randomness is a valuable resource to trade for computation efficiency. For example, there exist problems that can be solved efficiently by randomized algorithms but not deterministic algorithms. Second, there are applications, such as data mining, compressive sensing, etc., that may not be sensitive to minor random fluctuations. Third, devices at their quantum foundation or systems in their biological nature are intrinsically probabilistic. We aim at providing a formalism for the analysis and verification of probabilistic design. Notice that in the design automation process, verification is essential to the entire design flow. For example, equivalence checking [50, 74] should be applied to validate the correctness of synthesized circuits. Hence, establishing a verification framework is a crucial step in automated synthesis of probabilistic design.

Probabilistic behavior of a design has also been studied along the research of circuit reliability analysis, which focuses on analyzing the robustness of a circuit against

DOI

permanent defects or transient faults. The reliability of a circuit is characterized by the probability of the occurrence of an error at the primary outputs. Therefore, the study of circuit reliability is very related to the evaluation of probabilistic design.

Classical approaches to reliability analysis apply fault injection and Monte Carlo simulation [78]. Symbolic analysis methods exploiting mathematical tools, such as Markov random fields [4], probability transfer matrices [49], Bayesian networks [92], and algebraic decision diagrams [76], have also been investigated. Choudhury and Mohanram [20] propose three accurate and scalable algorithms to address the scalability issue of symbolic methods. However, prior methods for circuit reliability analysis are inadequate to probabilistic circuits for the following two reasons. First, most prior approaches assume single-gate failures. This assumption makes prior methods inapplicable to probabilistic design, where multiple probabilistic gates may be commonly present. Second, most prior efforts consider only the average error rate of a design. The necessity of analyzing the maximum error rate comes from the increasing demand of safety-centric systems, e.g., utilized in health care or automotive industries [60, 61]. However, its computational scalability is much limited due to the underlying symbolic modeling via Bayesian network [44].

DOI

2.2 Stochastic Boolean satisfiability

SSAT [63, 66] is first formulated by Papadimitriou and interpreted as *games against nature* [86]. It lies in the same PSPACE-complete [102] complexity class as QBF.

Exploiting randomized quantifiers, SSAT is capable of modeling a variety of computational problems inherent with uncertainty [41], such as probabilistic planning [51, 62], Bayesian-network inference [3, 24, 30, 44], and trust management [66]. Recently, the quantitative information-flow analysis for software security is also formulated as an E-MAJSAT formula [35], and bi-directional polynomial-time reductions between SSAT and POMDP are established [94].

A number of SSAT solvers have been developed. Among the prior efforts made to approach SSAT, most of them are based on Davis-Putnam-Logemann-Loveland (DPLL) search [28]. For example, solver `MAXPLAN` [68] encodes a conformant planning problem as an E-MAJSAT formula and improves the solving efficiency by pure variables, unit propagation, and subproblem memorization; solver `ZANDER` [69] deals with partially observable probabilistic planning by formulating the problem as a general SSAT formula and incorporates several threshold-pruning heuristics to reduce the search space; solver `DC-SSAT` [67] relies on a divide-and-conquer approach to break an SSAT formula into many subproblems and handles them separately to speedup the solving of a general SSAT formula. Approximate solving [65] and resolution rules [103] for SSAT have also been addressed. Techniques from *knowledge compi-*

DOI

lation have also been exploited to solve E-MAJSAT formulas. Solver `ComPlan` [42] compiles the matrix of an E-MAJSAT formula into its *deterministic, decomposable negation normal form* (d-DNNF) [25, 26], and performs a branch-and-bound search. It is further improved by an enhanced bound computation method [89].

2.3 Model counting

Model-counting [39] algorithms can be classified into two categories: exact counting and approximate counting. The former adopts DPLL-based search with additional techniques, such as component analysis and caching, to improve the counting efficiency[95, 96]. The latter takes a different strategy, aiming at providing lower and/or upper bounds with guarantee on confidence level. Ideas from statistics [16] have been adopted to increase the capacity limit of model counting.

There are many variants of model counting. For example, weighted model counting asks to aggregate the weight of every satisfying assignment. It has been widely adopted in probabilistic inference [18, 97]. *Projected model counting* [2] computes the numbers of satisfying assignments projected on a subset of original variables. *Maximum model counting* [35] finds an assignment to a subset of variables in a formula such that the number of satisfying assignments of the residual formula cofactored with the assignment is maximized.

The above variants of model counting can be expressed via SSAT, because

DOI

Table 2.1: Model-counting variants and their corresponding SSAT formulas

Model-counting variant	SSAT encoding
Unweighted	$\text{ꓤ}^{0.5}x_1, \ldots, \text{ꓤ}^{0.5}x_n.\phi(x_1, \ldots, x_n)$
Weighted	$\text{ꓤ}^{p_1}x_1, \ldots, \text{ꓤ}^{p_n}x_n.\phi(x_1, \ldots, x_n)$
Projected	$\text{ꓤ}^{0.5}x_1, \ldots, \text{ꓤ}^{0.5}x_n, \exists y_1, \ldots, \exists y_m.\phi(x_1, \ldots, x_n, y_1, \ldots, y_m)$
Maximum	$\exists x_1, \ldots, \exists x_n, \text{ꓤ}^{0.5}y_1, \ldots, \text{ꓤ}^{0.5}y_m.\phi(x_1, \ldots, x_n, y_1, \ldots, y_m)$
Projected weighted	$\text{ꓤ}^{p_1}x_1, \ldots, \text{ꓤ}^{p_n}x_n, \exists y_1, \ldots, \exists y_m.\phi(x_1, \ldots, x_n, y_1, \ldots, y_m)$
Maximum weighted	$\exists x_1, \ldots, \exists x_n, \text{ꓤ}^{p_1}y_1, \ldots, \text{ꓤ}^{p_m}y_m.\phi(x_1, \ldots, x_n, y_1, \ldots, y_m)$

the randomized quantifiers of SSAT essentially aggregate the results from different branches with weights. Table 2.1 shows the variants of model-counting problems and their respective SSAT encodings. Note that projected weighted model counting and maximum weighted model counting are equivalent to random-exist quantified SSAT and exist-random quantified SSAT, respectively.

The latest developments of model counting can be found in the report [34] of the 2020 Model Counting Competition.

DOI

Chapter 3

Background

In this chapter, we provide background knowledge that is commonly used across this dissertation. Preliminaries specific to each chapter will be introduced later when they are needed. Symbols used in this dissertation are summarized in Table 3.1.

3.1 Propositional logic

We denote Boolean constants FALSE and TRUE by symbols $\bot$ and $\top$, respectively. In arithmetic expressions, $\bot$ is interpreted as integer 0 and $\top$ as integer 1. A variable x that takes values from the Boolean domain $\mathbb{B} = \{\bot, \top\}$ is called a Boolean variable. A *literal* is a variable itself (a *positive* literal) or the negation of a variable (a *negative* literal). For a literal l, let $\texttt{var}(l)$ denote the variable of l. Boolean

DOI

connectives $\neg, \vee, \wedge, \rightarrow, \equiv$ are used under their conventional semantics. Over a finite set V of Boolean variables, we define a *well-formed formula* ϕ with the following Backus-Naur-form (BNF) grammar:

$$\phi ::= x \in V | \neg\phi | (\phi \vee \phi) | (\phi \wedge \phi) | (\phi \rightarrow \phi) | (\phi \equiv \phi). \tag{3.1}$$

Given a well-formed formula ϕ, let $\texttt{vars}(\phi)$ denote the set of Boolean variables appearing in ϕ. In the following, a variable is Boolean if not otherwise specified. We shall consider well-formed formulas only and refer to them as *Boolean formulas*.

3.1.1 Conjunctive and disjunctive normal form

Among various representations of a Boolean formula, we are particularly interested in normal-form representations because their simplicity allows efficient analyses.

A Boolean formula is in *conjunctive normal form* (CNF) if it is a conjunction of *clauses*, where a clause is a disjunction of literals. A Boolean formula is in *disjunctive normal form* (DNF) if it is a disjunction of *cubes*, where a cube is a conjunction of literals. A variable x is said to be *pure* in a formula if its appearances in the formula are all positive literals or negative literals. We alternatively treat a clause or a cube as a set of literals, and a CNF (resp. DNF) formula as a set of clauses (resp. cubes). In the rest of the dissertation, a Boolean formula is assumed to be given in CNF if not otherwise specified.

DOI

3.1.2 Boolean satisfiability

An *assignment* τ over a variable set V is a mapping from V to $\mathbb{B}$. We denote the set of all assignments over V by $\mathcal{A}(V)$. Given a Boolean formula ϕ, an assignment τ over $\texttt{vars}(\phi)$ is called a *complete* assignment for ϕ. If τ is over a proper subset of $\texttt{vars}(\phi)$, it is called a *partial* assignment. The resultant formula of ϕ induced by an assignment τ over a variable set V, denoted as $\phi|_\tau$, is obtained via substituting the occurrences of every $x \in V$ in ϕ with its assigned value $\tau(x)$. Such substitution is called *cofactoring* ϕ with τ. If $V = \{x\}$, we write $\phi|_x$ (resp. $\phi|_{\neg x}$) to denote the resultant formula of ϕ under an assignment that maps x to $\top$ (resp. $\bot$), and call this formula the *positive* (resp. *negative*) *cofactor* of ϕ with respect to variable x.

A complete assignment τ *satisfies* ϕ, denoted as $\tau \models \phi$, if $\phi|_\tau = \top$. Such complete assignment τ is called a *satisfying complete assignment* for ϕ. On the other hand, if $\phi|_\tau = \bot$, τ is called an *unsatisfying complete assignment*. Similarly, a partial assignment τ^+ over $X \subset \texttt{vars}(\phi)$ is called a *satisfying* (resp. an *unsatisfying*) *partial assignment* for ϕ if for some (resp. every) assignment μ over $\texttt{vars}(\phi) \setminus X$, ϕ valuates to $\top$ (resp. $\bot$) under the complete assignment that combines τ and μ. We alternatively represent an assignment τ for ϕ as a cube. A cube is called a *minterm* of formula ϕ when it corresponds to a complete assignment over $\texttt{vars}(\phi)$. Given two Boolean formulas ϕ_1 and ϕ_2 over a same set V of variables, we write $\phi_1 \models \phi_2$ if the following condition holds: $\forall \tau \in \mathcal{A}(V).\tau \models \phi_1 \rightarrow \tau \models \phi_2$.

DOI

A Boolean formula ϕ is *satisfiable* if it has a satisfying complete assignment. Otherwise, ϕ is *unsatisfiable*. A Boolean formula ϕ is a *tautology* if the following condition holds: $\forall \tau \in \mathcal{A}(\texttt{vars}(\phi)).\tau \models \phi$. The Boolean satisfiability problem asks to decide whether a Boolean formula is satisfiable or not. It is a well-known NP-complete [23] problem. We write $\texttt{SAT}(\phi)$ (resp. $\texttt{UNSAT}(\phi)$) to indicate ϕ is satisfiable (resp. unsatisfiable). A satisfying complete assignment of ϕ is also called a *model* of ϕ, which is denoted by ϕ.model.

A set $X_d \subseteq \texttt{vars}(\phi)$ is a *base set* for ϕ if for any (partial) assignment τ^+ over X_d, there exists at most one assignment μ over $\texttt{vars}(\phi) \setminus X_d$ such that ϕ is satisfied by the combined assignment of τ^+ and μ over $\texttt{vars}(\phi)$. Observe that, given any Boolean formula ϕ, a base set must exist ($\texttt{vars}(\phi)$ is a trivial base set of ϕ) but may not be unique. Let τ^+ be an assignment over a base set $X_d \subseteq \texttt{vars}(\phi)$. If there exists an assignment μ over $\texttt{vars}(\phi) \setminus X_d$ such that the combined assignment ν satisfies ϕ, then we say that ϕ is satisfiable under τ^+ and write $\tau^+ \models \phi$ to mean $\nu \models \phi$. If there does not exist such an assignment μ over $\texttt{vars}(\phi) \setminus X_d$, then ϕ is unsatisfiable under τ^+, denoted by $\tau^+ \not\models \phi$.

An n-variable *Boolean function* is a mapping from $\mathbb{B}^n$ to $\mathbb{B}$. Note that a Boolean formula ϕ induces a Boolean function with a domain $\mathcal{A}(\texttt{vars}(\phi))$. We shall not distinguish between a Boolean formula and its induced Boolean function.

DOI

3.2 Model counting

3.2.1 Exact/Approximate model counting

The *model counting* [39] problem asks to find the number of the satisfying assignments of a Boolean formula ϕ. Model-counting algorithms can be classified into two categories: *exact* algorithms and *approximate* algorithms. The former computes the exact number of the satisfying assignments of a formula; the latter computes upper or lower bounds of the number of the satisfying assignments with a confidence level.

3.2.2 Weighted model counting

The *weighted model counting* problem asks to compute the weight of a formula ϕ given a weighting function $\omega : \texttt{vars}(\phi) \mapsto [0, 1]$. The weight of a positive literal x (resp. a negative literal $\neg x$) is defined to be $\omega(x)$ (resp. $1 - \omega(x)$). The weight of an assignment τ, denoted as $\omega(\tau)$, equals the product of the weights of its individual literals. The weight of the formula ϕ, denoted as $\omega(\phi)$, is the summation of the weights of its satisfying assignments.

DOI

3.3 Stochastic Boolean satisfiability

An SSAT formula Φ over variables $\{x_1, \ldots, x_n\}$ has the form: $Q_1 x_1, \ldots, Q_n x_n . \phi$, where each $Q_i \in \{\exists, \text{ꓤ}^p\}$ and ϕ is quantifier-free. Symbol $\exists$ denotes an existential quantifier with its conventional semantics. Symbol ꓤ^p denotes a randomized quantifier [86], which requires the quantified variable to evaluate to $\top$ with probability $p \in [0, 1]$. Given an SSAT formula Φ, the quantification structure $Q_1 x_1, \ldots, Q_n x_n$ is called the *prefix*, and the quantifier-free Boolean formula ϕ is called the *matrix*.

Let x be the outermost variable in the prefix of an SSAT formula Φ. The satisfying probability of Φ, denoted by $\Pr[\Phi]$, is defined by the following four rules:

a) $\Pr[\top] = 1$,

b) $\Pr[\bot] = 0$,

c) $\Pr[\Phi] = \max\{\Pr[\Phi|_{\neg x}], \Pr[\Phi|_x]\}$, if x is existentially quantified,

d) $\Pr[\Phi] = (1 - p)\Pr[\Phi|_{\neg x}] + p\Pr[\Phi|_x]$, if x is randomly quantified by ꓤ^p,

where $\Phi|_{\neg x}$ and $\Phi|_x$ denote the SSAT formulas obtained by eliminating the outermost quantifier of x via substituting the value of x in the matrix with $\bot$ and $\top$, respectively.

The *decision version* of SSAT is stated as follows. Given an SSAT formula Φ and a threshold $\theta \in [0, 1]$, decide whether $\Pr[\Phi] \geq \theta$. On the other hand, the

DOI

optimization version asks to compute the exact value of $\Pr[\Phi]$. The decision version of SSAT is PSPACE-complete [86].

DOI

Table 3.1: Summary of the symbols used in the dissertation

Symbol	Description
$\mathbb{B}$	The Boolean domain $\{\bot, \top\}$
x	A Boolean variable
l	A literal (a variable or its negation)
$\texttt{var}(l)$	The variable of l
τ	An assignment (a mapping from a variable set to $\mathbb{B}$)
$\tau(x)$	The assigned value of x
$\mathcal{A}(V)$	The set of all assignments over variable set V
ϕ	A quantifier-free formula
$\texttt{vars}(\phi)$	The set of variables appearing in ϕ
$\tau \models \phi$	τ satisfies ϕ
$\phi_1 \models \phi_2$	Every satisfying assignment of ϕ_1 also satisfies ϕ_2
$\texttt{SAT}(\phi), \texttt{UNSAT}(\phi)$	ϕ is (un)satisfiable
ϕ.model	A satisfying assignment of ϕ
$\phi\|_x, \phi\|_{\neg x}$	Positive and negative cofactors of ϕ w.r.t. x
$\phi\|_\tau$	The resultant formula after cofactoring ϕ with τ
C	A clause (a disjunction of literals)
Φ	A quantified formula

DOI

Chapter 4

Probabilistic Design Evaluation

In this chapter, we propose a general formulation for the evaluation and verification of probabilistic design. We establish the connection between the proposed formulation and SSAT, weighted model counting, and probabilistic model checking. Moreover, a new SSAT algorithm based on *binary decision diagram* (BDD) is proposed. Most content in this chapter is based on our journal paper [54] published in IEEE Transactions on Computers.

DOI

4.1 Preliminaries

4.1.1 Boolean network

A *(combinational) Boolean network* is a directed acyclic graph $G = (V, E)$, with a set V of vertices and a set $E \subseteq V \times V$ of edges. Two non-empty disjoint subsets V_I and V_O of V are identified: a vertex $v \in V_I$ (resp. V_O) is referred to as a *primary input* (PI) (resp. *primary output* (PO)). Each vertex $v \in V$ is associated with a Boolean variable b_v. Each vertex $v \in V \setminus V_I$ is associated with a Boolean function f_v. An edge $(u, v) \in E$ indicates f_v refers to b_u as an input variable; u is called a *fanin* of v, and v a *fanout* of u. The valuation of the Boolean variable b_v of vertex v is as follows: if v is a PI, b_v is given by external signals; otherwise, b_v equals the value of f_v. To ease readability, we will not distinguish a vertex v and its corresponding Boolean variable b_v. We will simply denote b_v with v.

Note that a Boolean network can be converted in linear time to a CNF formula through Tseitin transformation [105]. Consider a Boolean network G, and let X denote the set of PI variables of G. During Tseitin transformation, new variables will be introduced for every vertex $v \in V \setminus V_I$. Let Z denote the set of these fresh variables. The resultant formula $\phi_G(X, Z)$ obtained from Tseitin transformation encodes the behavior of the Boolean network G. Observe that X is a base set for ϕ_G. This is because once the PI variables are decided by an assignment τ^+ over X, the values for the other variables will be propagated according to the behavior of

DOI

the Boolean network. Therefore, at most one assignment μ over Z (the one with the consistent variable evaluation to the Boolean network) is able to satisfy ϕ_G.

4.1.2 Probability and random variables

To characterize the behavior of a probabilistic design, we take advantage of Bernoulli random variables. In the following, we provide basic definitions of random variables.

Consider an experiment with a sample space S and a probability measure $\Pr[\cdot]$. A *random variable* X is a mapping from an outcome in S to a real number. The *probability mass function* (PMF) P_X of X is defined by $P_X(x) = \Pr[\{s \mid X(s) = x\}]$.

A random variable X is called a *Bernoulli*(p) *random variable* with parameter $p \in [0, 1]$, denoted by $X \sim$ *Bernoulli*(p), if the PMF of X has the form:

$$P_X(x) = \begin{cases} p, & \text{if } x = 1, \\ 1 - p, & \text{else if } x = 0, \\ 0, & \text{otherwise.} \end{cases}$$

Note that a Bernoulli random variable maps every outcome in a sample space to either 0 or 1. Therefore, it is suitable to characterize experiments with binary outcomes.

For a wire (an edge) of a circuit (Boolean network), its value is either TRUE or FALSE. A Bernoulli random variable in this context maps TRUE and FALSE to real numbers 1 and 0, respectively. The corresponding parameter p of the random

DOI

variable is the probability for the wire to valuate to TRUE. On the other hand, for a gate (a vertex) of a circuit, its operation has two outcomes: correct and erroneous. A Bernoulli random variable in this context maps erroneous and correct operations to real numbers 1 and 0, respectively. The corresponding parameter p of the random variable is the probability of erroneous operation, i.e., the error rate, of the gate.

4.2 Modeling probabilistic design

To verify a deterministic design, properties are often asserted at the primary outputs of a circuit to examine whether there exists an assignment to the primary inputs that falsifies some of the properties. If there exists such an assignment, a counterexample to a property is found.

However, when it comes to probabilistic design, the same approach is not fully adequate. Since a probabilistic circuit could produce different output responses for the same input stimulus, computing the probability of property violation is more meaningful than searching for a counterexample. Hence we pose the following question: *Given a probabilistic circuit and a property to be verified, what is the average or maximum probability for the property to be violated?* We model a probabilistic design as a probabilistic Boolean network and formalize *probabilistic property evaluation* (PPE) to answer the question. While the evaluation of combinational probabilistic design is of our primary interest, the proposed framework is also ex-

DOI

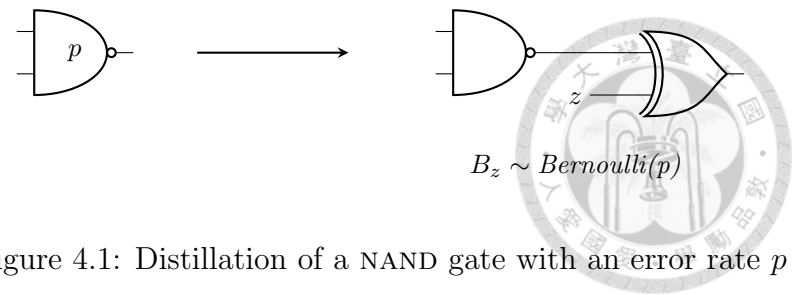


Figure 4.1: Distillation of a NAND gate with an error rate p

tensible to sequential probabilistic design.

4.2.1 Probabilistic Boolean network

We define a probabilistic Boolean network to model circuits with logic gates that exhibit probabilistic behavior. A *probabilistic Boolean network* (PBN) is a Boolean network $G = (V, E)$ with random variables annotated to its vertices. The probabilistic behavior of a PI $v \in V_I$ is modeled by a Bernoulli random variable $B_v \sim$ *Bernoulli*(p_v) with $p_v = \Pr[v = \top]$. The probabilistic behavior of the output of a vertex $v \in V \setminus V_I$ is modeled with a Bernoulli random variable $B_v \sim$ *Bernoulli*(p_v) with p_v corresponding to the error rate of v.

In general, two random variables B_u and B_v for $u, v \in V$ and $u \neq v$ can be dependent, and their joint distribution has to be considered. In this paper, we first focus on the simplified situation where the random variables of vertices are mutually independent, and refer to a PBN whose random variables are mutually independent as an *independent PBN*. We will show how to extend the proposed framework to a PBN with mutually dependent random variables later.

DOI

Given a PBN $G = (V, E)$, without loss of generality, we standardize G by converting it to a *standardized PBN* (SPBN) $G' = (V', E')$ with the *distillation operation* depicted in Fig. 4.1, using an erroneous NAND gate as an example. For each node $v \in V \setminus V_I$ of G with an error rate p_v, its probabilistic behavior is distilled into an error source modeled by an auxiliary Boolean input z with $B_z \sim \mathit{Bernoulli}(p_v)$ for $\Pr[z = \top] = p_v$. Moreover, an XOR gate is used to conditionally invert the output of the error-free node v if and only if z valuates to $\top$. Note that an auxiliary input differs from a primary input in that the error rate p_z of an auxiliary input is fixed with respect to the characteristics of a probabilistic node rather than determined by the environmental input behavior. In the following, we let V_Z be the set of auxiliary inputs (AIs) of an SPBN.

4.2.2 Probabilistic property evaluation

To formally reason about probabilistic design, we formulate *probabilistic property evaluation* (PPE) by capturing the degree of property violation with probability. We distinguish between *input assignment* and *parameter assignment* in PPE. An *input assignment* assigns a truth value to a Boolean variable v for every PI. On the other hand, a *parameter assignment* specifies a probability p_v for a PI v to be TRUE. Note that the probability of an error source $z \in V_Z$ is determined by the underlying PBN and needs not be assigned.

To analyze a probabilistic design, we define *signal probability* as follows.

DOI

Definition 4.1 (Signal Probability (parameterized)). *Given an SPBN $G = (V, E)$ and a parameter assignment $\pi : V_I \mapsto [0, 1]$, the signal probability or satisfying probability of a node $v \in V$ with respect to the parameter assignment π is* $\Pr[v = \top]$ *under π.*

It is natural to ask under what parameter assignment the signal probability of some node is maximized. It corresponds to the following definition.

Definition 4.2 (Signal Probability (maximized)). *Given an SPBN $G = (V, E)$, the maximum signal probability or maximum satisfying probability of a node $v \in V$ is* $\Pr[v = \top]$ *maximized over all parameter assignments.*

Notice that, given a parameter assignment to the PIs of an SPBN $G = (V, E)$, we could associate a random variable $R_v \sim \mathit{Bernoulli}(p)$ with each node $v \in V$ such that $p = \Pr[v = \top]$ under π. Those random variables can be mutually dependent (even for an SPBN derived from an independent PBN) due to the reconvergent paths in the Boolean network.

According to Definition 4.2, we have the following proposition on parameter assignments that maximize signal probability.

Proposition 4.1. *Given an SPBN $G = (V, E)$ and an arbitrary $v \in V$, there exists a parameter assignment π that maximizes the signal probability of v such that $\pi(u)$ equals either probability 0 or 1 for any $u \in V_I$.*

Proof. Assume there exists an optimizing parameter assignment π not in such a

DOI

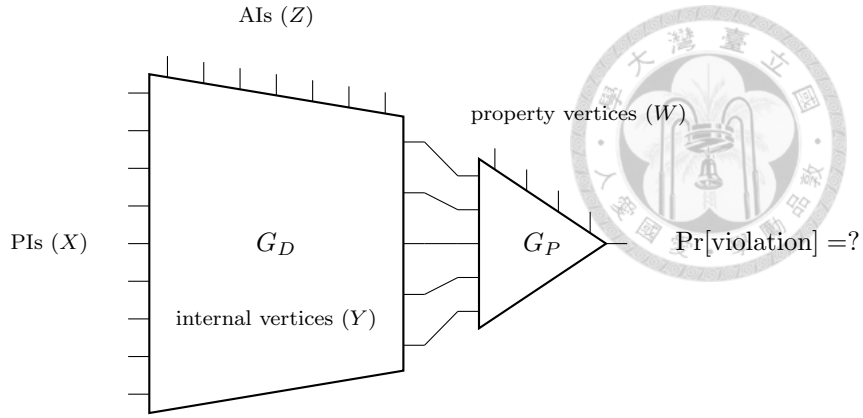


Figure 4.2: A miter SPBN for probabilistic property evaluation

form. That is, there exists some $u \in V_I$ such that $0 < \pi(u) < 1$. Denote the signal probabilities of v under $\pi(u) = 0$ and $\pi(u) = 1$ by p_0 and p_1, respectively. Note that $\Pr[v = \top]$ under π equals $(1-\pi(u)) \times p_0 + \pi(u) \times p_1$, and $\min\{p_0, p_1\} \leq \Pr[v = \top] \leq \max\{p_0, p_1\}$. If $p_0 \neq p_1$, there is a contradiction since $\Pr[v = \top] < \max\{p_0, p_1\}$, and π is not an optimizing assignment. If $p_0 = p_1$, W.L.O.G., set $\pi(u) = 0$. So the optimizing assignment must be in the stated form. ■

To formulate the PPE problem, we construct a *miter SPBN* as shown in Fig. 4.2. Given a *design SPBN* G_D modeling a probabilistic circuit and a *property SPBN* G_P specifying a property under verification, a miter SPBN G_M is built by connecting relevant primary outputs of G_D to the primary inputs of G_P. The property SPBN is used to monitor the design SPBN. Let X be the set of PIs of G_D, Z be the set of AIs of G_D, W be a subset of PIs of G_P, and Y be the set of all other vertices. Note that G_P may have additional inputs W to enrich the expressiveness of property. For example, they can be used to prioritize the primary outputs of G_D. Based on signal

DOI

probability, two versions of the PPE problem are defined below.

Definition 4.3 (PPE (maximized)). *Given a miter SPBN G_M with the vertex sets X, Y, Z, W defined as above, the maximum probabilistic property evaluation (MPPE) problem asks to find the maximum satisfying probability of the output of G_M. That is, the signal probability of the miter output corresponds to the maximum probability of property violation under all parameter assignments.*

Definition 4.4 (PPE (parameterized)). *Given a miter SPBN G_M with the vertex sets X, Y, Z, W defined as above and a parameter assignment $\pi : X \mapsto [0, 1]$, the probabilistic property evaluation (PPE) problem asks to find the satisfying probability of the output of G_M. That is, the signal probability of the miter output corresponds to the probability of property violation under the given parameter assignment.*

4.2.3 Extension to sequential probabilistic design

Although the above PPE and MPPE frameworks mainly focus on combinational design, they are extensible to analyze sequential design by *circuit unrolling* [21], similar to the soft-error reliability analysis for sequential circuits [75]. For example, to find the probability of property violation after T clocks of execution, the sequential circuit is *unrolled* into a combinational circuit by connecting T copies of the combinational block of the sequential circuit to mimic the state transitions in T time frames. For simplicity, the occurrences of errors among different time frames are assumed to be temporally independent, i.e., the random variables governing the

DOI

probabilistic behavior of errors among different time frames are mutually independent. After unrolling, the proposed PPE and MPPE frameworks can be applied to the effective combinational circuit and analyze the satisfying probability of the output at the Tth time frame. The result corresponds to the probability of property violation of the sequential design after T clocks of execution.

4.3 Solving probabilistic property evaluation

We propose different solutions to the MPPE and PPE problems. For the former, we resort to SSAT solving. For the latter, in addition to SSAT solving, we present solutions based on signal probability calculation, weighted model counting, and probabilistic model checking.

4.3.1 Solving MPPE and PPE via SSAT

Note that the entire miter SPBN can be directly converted to a CNF formula by Tseitin transformation [105] since all vertices except for those in X, Z, W are error-free after the standardization. In the following, let ϕ_M be the CNF formula converted from the miter SPBN G_M, which contains vertex sets $X = \{x_1, \ldots, x_n\}$, $Y = \{y_1, \ldots, y_m\}$, $Z = \{z_1, \ldots, z_l\}$, and $W = \{w_1, \ldots, w_q\}$ as shown in Fig. 4.2. Observe that $X \cup Z \cup W$ is a base set for ϕ_M. Given a parameter assignment π to X, we define the corresponding weighting function $\omega : X \cup Z \cup W \mapsto [0, 1]$ for ϕ_M as

DOI

$\omega(x_i) = \pi(x_i)$, $\omega(y_j) = p_{y_j}$, and $\omega(w_k) = p_{w_k}$ for all $x_i \in X$, $y_j \in Y$, and $w_k \in W$. The weight assignment ω will be used throughout our discussion.

Theorem 4.1. *The probabilistic property evaluation (PPE) of ϕ_M under a parameter assignment π can be expressed by the following SSAT formula $\Phi_{\text{PPE}}(\pi)$:*

$$\text{Ꙗ}^{\pi(x_1)} x_1, \ldots, \text{Ꙗ}^{\pi(x_n)} x_n, \text{Ꙗ}^{p_{z_1}} z_1, \ldots, \text{Ꙗ}^{p_{z_l}} z_l, \text{Ꙗ}^{p_{w_1}} w_1, \ldots, \text{Ꙗ}^{p_{w_q}} w_q, \exists y_1, \ldots, \exists y_m . \phi_M. \tag{4.1}$$

Proof. Let A be the event $\phi_M = \top$ and $\Lambda = \mathcal{A}(X \cup Z \cup W)$. By the law of total probability, $\Pr[A] = \sum_{\tau \in \Lambda} \Pr[\tau] \Pr[A \mid \tau]$, where $\Pr[\tau] = \omega(\tau)$ (ω is the weighting function defined previously) and $\Pr[A \mid \tau]$ is the conditional probability of event A under the assignment τ. Notice that $\Pr[A \mid \tau] = \phi_M|_\tau$ since $X \cup Z \cup W$ is a base set for ϕ_M. As a result, $\Pr[A] = \sum_{\tau \in \Lambda} \omega(\tau) \phi_M|_\tau$, which equals the satisfying probability of the SSAT formula $\Phi_{\text{PPE}}(\pi)$. ■

Theorem 4.2. *The maximum probabilistic property evaluation (MPPE) of ϕ_M can be expressed by the following SSAT formula Φ_{MPPE}:*

$$\exists x_1, \ldots, \exists x_n, \text{Ꙗ}^{p_{z_1}} z_1, \ldots, \text{Ꙗ}^{p_{z_l}} z_l, \text{Ꙗ}^{p_{w_1}} w_1, \ldots, \text{Ꙗ}^{p_{w_q}} w_q, \exists y_1, \ldots, \exists y_m . \phi_M. \tag{4.2}$$

Proof. By the same argument in the proof of Theorem 4.1, given an assignment τ_X over X, the SSAT formula:

$$\Phi_{\text{MPPE}}(\tau_X) = \text{Ꙗ}^{p_{z_1}} z_1, \ldots, \text{Ꙗ}^{p_{z_l}} z_l, \text{Ꙗ}^{p_{w_1}} w_1, \ldots, \text{Ꙗ}^{p_{w_q}} w_q, \exists y_1, \ldots, \exists y_m . \phi_M|_{\tau_X}$$

computes the satisfying probability of the miter under the assignment τ_X. According to the SSAT semantics, the outermost existential quantification of primary inputs

DOI

Algorithm 1 BDD-based SSAT solving: `BddSsatSolve`

Input: $\Phi = Q_1 v_1, \ldots, Q_n v_n.\phi$

Output: $\Pr[\Phi]$

1: $N :=$ `BuildReducedOrderedBdd`$(\phi, (v_1, \ldots, v_n))$

2: $Q := Q_1 v_1, \ldots, Q_n v_n$

3: **return** `BddSsatRecur`(N, Q)

X ensures to find an optimum assignment τ_X^* such that the satisfying probability of $\Phi_{\text{MPPE}}(\tau_X^*)$ is maximized. Hence the SSAT formula Φ_{MPPE} computes the maximum satisfying probability of the miter. ■

Note that the only difference between Φ_{MPPE} and Φ_{PPE} lies in the quantification for the primary inputs X. Although SSAT provides a convenient language for expressing both MPPE and PPE problems, its solvers to date remain immature to handle formulas of practical sizes in our considered application. One of the main inefficiencies can be attributed to representing ϕ_M in CNF, which results in the additional quantification of the intermediate circuit variables $Y = \{y_1, \ldots, y_m\}$. It motivates the development of a new SSAT solver as we present below.

BDD-based SSAT solving

We propose a BDD-based solver to enhance the scalability of SSAT solving. The procedure `BddSsatSolve` is outlined in Alg. 1, which takes as input an SSAT formula $\Phi = Q_1 v_1, \ldots, Q_n v_n.\phi$. At line 1, a *reduced ordered BDD* (ROBDD) of ϕ is built

DOI

Algorithm 2 The recursive step of `BddSsatSolve`: `BddSsatRecur`

Input: An ROBDD node N and a prefix Q

Output: $\Pr[N = \top]$ under Q

1: **if** (N is a terminal node) **then**

2: **return** N.sp

3: **end if**

4: **if** (N.visited = FALSE) **then**

5: **if** ($Q(N.\text{var}) = \exists^p$) **then**

6: $N.\text{sp} := (1-p) \cdot \texttt{BddSsatRecur}(N.\text{else}, Q) + p \cdot \texttt{BddSsatRecur}(N.\text{then}, Q)$

7: **else**

8: $N.\text{sp} := \max\{\texttt{BddSsatRecur}(N.\text{else}, Q), \texttt{BddSsatRecur}(N.\text{then}, Q)\}$

9: **end if**

10: N.visited := TRUE

11: **end if**

12: **return** N.sp

with a variable ordering following the quantification order. At line 3, a recursive procedure `BddSsatRecur`, sketched in Alg. 2, is called to calculate the satisfying probability of Φ. In the pseudo code, for an ROBDD node N, N.then and N.else denote its then- and else-child, respectively; N.value equals 0 (resp. 1) if N is a 0-terminal (resp. 1-terminal) node; N.visited is a flag initialized to FALSE and records whether N has been processed; N.var and N.sp denote the control variable and the satisfying probability of node N, respectively. In Alg. 2, lines 1 to 2 implement

DOI

the first and second computation rules of SSAT in Section 3.3; Line 6 and line 8 implement the third and fourth rules corresponding to the random and existential quantification of the variable, respectively.

Note that `BddSsatSolve` runs in time linear to the number of BDD nodes (as each node is processed only once). Therefore the computation complexity is dominated by constructing the ROBDD of ϕ. Note also that if the outermost variables in the quantification order are existentially quantified, e.g., those in Φ_{MPPE} of Theorem 4.2, the corresponding assignments to the existentially quantified variables to maximize the satisfying probability of ϕ can be obtained. Specifically, the assignment to an outermost existential variable N.var can be derived by recording which of `BddSsatRecur`$(N.\text{else}, Q)$ and `BddSsatRecur`$(N.\text{then}, Q)$ contributes to the maximum probability in line 8 of Alg. 2.

Signal Probability with BDD-based SSAT

We exploit the developed BDD-based SSAT solver to compute signal probabilities as defined in Definitions 4.1 and 4.2. Given an SPBN $G = (V, E)$, the satisfying probability of any $v \in V$ can be obtained by first encoding the problem into an SSAT instance and then solving the SSAT formula by `BddSsatSolve`.

Notice that formula ϕ needs not be represented in CNF for `BddSsatSolve`. In the application of circuit verification, formula ϕ can be directly input to `BddSsatSolve`

DOI

as a circuit. Without Tseitin transformation from circuit to CNF formula, the algorithm avoids introducing extra variables. Consequently, the innermost existential quantification $\exists y_1, \ldots, \exists y_m$ in Eq. (4.1) of Theorem 4.1 and Eq. (4.2) of Theorem 4.2 is removed. The utilization of circuit structures makes the calculation of signal probability (and therefore the computation of PPE and MPPE) on an SPBN more efficient and scalable using the proposed BDD-based SSAT solver. Empirical evidence suggests our proposed SSAT solver outperforms and is much scalable than other CNF-based SSAT solvers.

We note that the signal probability calculation via BDD is used for power estimation of integrated circuits [81]. As we formulate PPE as a signal probability problem on SPBN, any prior method for signal probability calculation can be used to solve PPE. However, it is worth emphasizing that prior methods for signal probability calculation, such as Monte Carlo simulation, cannot solve MPPE. The BDD-based method has its unique value over other previous endeavors for signal probability calculation because of its generality of solving both PPE and MPPE. On the other hand, since we established the connections of MPPE and PPE to SSAT formulations, SSAT solvers can also be applied to calculate the maximum signal probability as well as signal probability.

BDD is a well-studied data structure, but known for its memory limitation. Techniques have been highly developed in the 1990s to extend its scalability. Practical experience suggests that BDD-based computation remains competitive to other

DOI

formulations due to the fact that other tools, such as SSAT and model counting, are still in their early development. In our experiments over ISCAS and ITC benchmark suits, the BDD-based approach stands as the most scalable one over other formulations to be discussed.

Generalization to dependent random variables

The proposed BDD-based SSAT solver is advantageous over other SSAT solvers for its ability to handle randomly quantified variables whose probability distributions are mutually dependent. Note that according to the SSAT syntax, there is no support to describe the joint behavior among randomly quantified variables. That is, every randomly quantified variable acts independently of each other. This assumption limits the expressiveness of SSAT. In this paper, we combine our BDD-based SSAT solver with previous methods [70, 76] to represent joint probability distribution of random variables, and propose a novel SSAT solver that is capable of expressing mutual dependence among randomly quantified variables. A joint probability distribution of random variables is represented as an algebraic decision diagram (ADD) [70, 76]. After such an ADD representing joint probability distribution is constructed, the original BDD (which is also an ADD) of the Boolean formula ϕ is conjoined with the ADD. Finally, the value of the SSAT formula can be calculated by traversing the merged ADD similar to the prior independent counterpart.

Following the above strategy, now we explain how to extend the proposed PPE

DOI

and MPPE framework to approach PBNs whose random variables are mutually dependent. After the distillation operation, the mutually dependent random variables on erroneous vertices are converted to correlated AIs. Given the joint probability distribution of random variables, an ADD is built to represent the mutual dependence among AIs. Similarly, if PIs are correlated, another ADD can be built to describe their correlation. After multiplying the ADDs with the BDD of the circuit under evaluation, the average (PPE) or the maximum (MPPE) violating probability can be computed through traversing the product ADD. Therefore, the proposed PPE framework is generalized to dependent PBNs, whose random variables have mutual dependent probability distribution.

4.3.2 Solving PPE via weighted model counting

The following theorem states that an SSAT formula of the form in Theorem 4.1 is equivalent to a weighted model counting instance.

Theorem 4.3. *The SSAT formula:*

$$\Phi = \text{ꓤ}^{p_{x_1}} x_1, \ldots, \text{ꓤ}^{p_{x_n}} x_n, \exists y_1, \ldots, \exists y_m.\phi,$$

where $X = \{x_1, \ldots, x_n\}$ is a base set for ϕ, is equivalent to a weighted model counting instance of ϕ under a weighting function ω such that $\omega(x_i) = p_{x_i}$ for every $x_i \in X$.

Proof. Let A be the event that $\phi = \top$. According to Theorem 4.1, we have $\Pr[A] =$

DOI

$\sum_{\tau \in \mathcal{A}(X)} \omega(\tau)\phi|_\tau$. Since X is a base set for ϕ, $\Pr[A]$ can be simplified to $\sum_{\tau \models \phi} \omega(\tau)$, which is the weight of ϕ. ■

By the above theorem, weighted model counting is clearly applicable to PPE.

While exact model counting can be extended to the weighted version at little extra cost, more effort is required to achieve approximate weighted model counting [39]. To enhance the scalability of solving PPE via model counting, we show how to rewrite a weighted model counting instance into an equivalent unweighted formula. The rewriting procedure `WmcRewriting` is outlined in Alg. 3.

The following lemma explains the rationale behind `WmcRewriting`.

Lemma 4.4. *Let ϕ be a Boolean formula with a base set X_d and a weighting function ω over X_d. Given an arbitrary variable $x \in X_d$, we construct X'_d, ϕ', and ω' as follows. Let $X'_d = X_d \setminus \{x\} \cup \{y_x, z_x\}$ with y_x, z_x being newly introduced fresh variables. Let $\phi' = \phi \wedge ((inv \oplus x) \equiv (y_x \wedge z_x))$, where inv is either $\top$ or $\bot$ to determine the sign of x. If $inv = \bot$, let $\omega'(y_x)\omega'(z_x) = \omega(x)$; else, let $\omega'(y_x)\omega'(z_x) = 1 - \omega(x)$. For any other variable $v \in X'_d$, $\omega'(v) = \omega(v)$. After this construction, X'_d is a base set for ϕ', and $\omega'(\phi') = \omega(\phi)$.*

Proof. First, we show that X'_d is a base set for ϕ'. Clearly any assignment τ' over X'_d can be transformed into an assignment τ over X_d by substituting $\tau'(y_x), \tau'(z_x)$ into the formula $((inv \oplus x) \equiv y_x \wedge z_x)$ to derive the truth value of x. The fact that X_d being a base set for ϕ implies X'_d being a base set for ϕ'.

DOI

Algorithm 3 Formula rewriting for unweighted model counting: `WmcRewriting`

Input: A formula ϕ, a base set X_d for ϕ, a wt. func. ω s.t. $\forall x \in X_d.\omega(x) = \frac{k}{2^n}$

Output: A formula ϕ', a base set X'_d for ϕ', a wt. func. ω' s.t. $\forall x \in X'_d.\omega'(x) = \frac{1}{2}$

1: $\phi' := \phi, X'_d := X_d$

2: **for all** $(x \in X_d)$ **do**

3: $\quad var := x, wt := \omega(x)$

4: $\quad$ **while** $(wt \neq \frac{1}{2})$ **do**

5: $\quad\quad inv := \bot$

6: $\quad\quad$ **if** $(wt > \frac{1}{2})$ **then**

7: $\quad\quad\quad wt := 1 - wt, inv := \top$

8: $\quad\quad$ **end if**

9: $\quad\quad \phi' := \phi' \wedge ((inv \oplus var) \equiv (y_{var} \wedge z_{var}))$

10: $\quad\quad X'_d := X'_d \setminus \{var\} \cup \{y_{var}\}$

11: $\quad\quad \omega'(y_{var}) = \frac{1}{2}$

12: $\quad\quad var = z_{var}, wt = 2 \cdot wt$

13: $\quad$ **end while**

14: $\quad X'_d := X'_d \cup \{var\}, \omega'(var) = \frac{1}{2}$

15: **end for**

16: **return** (ϕ', X'_d, ω')

DOI

Second, we prove that $\omega'(\phi') = \omega(\phi)$. We only show the case for $inv = \bot$, since the other case can be established similarly. Consider any τ over X_d. There are two cases: $\tau(x) = 1$ and $\tau(x) = 0$. In both cases, we derive corresponding assignments τ' such that $\phi|_\tau = \phi'|_{\tau'}$ and $\omega(\tau) = \sum \omega'(\tau')$. In the first case $\tau(x) = 1$, the corresponding τ' over X'_d is obtained by assigning $(\tau'(y_x), \tau'(z_x))$ to $(1, 1)$ and $\tau'(v) = \tau(v)$ for any other $v \in X'_d$. Obviously $\phi|_\tau = \phi'|_{\tau'}$, and $\omega(\tau) = \omega'(\tau')$ because $\omega(x) = \omega'(y_x)\omega'(z_x)$. In the second case $\tau(x) = 0$, there are three possible assignments τ' over X'_d, namely $(\tau'(y_x), \tau'(z_x)) = (0, 0), (0, 1), (1, 0)$, and $\tau'(v) = \tau(v)$ for any other $v \in X'_d$. Denote the three assignments by $\tau'_1, \tau'_2, \tau'_3$. Note that $\phi|_\tau = \phi'|_{\tau'_1} = \phi'|_{\tau'_2} = \phi'|_{\tau'_2}$, and $\omega(\tau) = \omega'(\tau'_1) + \omega'(\tau'_2) + \omega'(\tau'_3)$.

The above analysis shows that any $\tau \models \phi$ can be transformed into one or multiple τ' such that $\tau' \models \phi'$ and $\omega(\tau) = \sum \omega'(\tau')$. As a result, $\omega'(\phi') = \omega(\phi)$. ■

Theorem 4.5. *Given a weighted model counting instance ϕ and a weighting function ω over a base set X_d of ϕ such that for any $x \in X_d$, $\omega(x)$ has the form of $k/2^n$ for some $n \in \mathbb{N}$, k an odd integer, and $k < 2^n$, `WmcRewriting` in Alg. 3 derives ϕ' and ω' over X'_d such that X'_d is a base set for ϕ', $\omega'(\phi') = \omega(\phi)$, and $\omega'(x) = \frac{1}{2}$ for every $x \in X'_d$.*

Proof. To show the correctness of `WmcRewriting`, we divide the task into two parts. First, we show that `WmcRewriting` always terminates. Second, we prove that when it terminates, the claimed properties hold.

DOI

Consider a variable $x \in X_d$ with $\omega(x) = \frac{k}{2^n}$. Observe that the loop invariant of the WHILE loop is $wt = \frac{h}{2^m}$, for some $m \in \mathbb{N}$, h an odd integer, and $h < 2^m$. Moreover, after an iteration, the denominator of wt will be halved, while the numerator remains an odd integer. As a result, wt equals $1/2$ after $n-1$ iterations and the WHILE loop terminates.

Inside the WHILE loop, the truth value of inv is decided by comparing wt with $\frac{1}{2}$, and the corresponding formula rewriting and weight assignments are made as described in Lemma 4.4. According to Lemma 4.4, X'_d is a base set for ϕ' and $\omega'(\phi') = \omega(\phi)$. Furthermore, $\omega'(x) = \frac{1}{2}$ for every $x \in X'_d$ as assigned in the algorithm. ■

By the above theorem, a weighted model counting instance whose weighting function is specialized as above can be rewritten and solved by any (either exact or approximate) unweighted model counting engine, since $\omega(\phi) = \omega'(\phi') = \sum_{\tau \models \phi'} \omega'(\tau) = \#\phi'/2^{|X'_d|}$, where $|X'_d|$ is the cardinality of the set X'_d. We remark that, given a variable x and its weight $p = k/2^n$, the cost, in terms of the number of added variables and clauses, of `WmcRewriting` is linear to n. In a way, n can be interpreted as the degree of precision if we attempt to apply `WmcRewriting` to approximate any arbitrary probability.

DOI

4.3.3 Solving PPE via probabilistic model checking

Probabilistic model checking (PMC) verifies stochastic systems modeled by the variants of Markov chains, e.g., discrete-time Markov chains (DTMCs), continuous-time Markov chains, and Markov decision processes, against properties specified in probabilistic temporal logics. We show how to calculate signal probability with the probabilistic model checker `PRISM` [52], which provides a high level modeling language to specify probabilistic systems.

We convert an SPBN to a DTMC and encode the calculation of signal probabilities by probabilistic computation tree logic (PCTL) [40]. To illustrate, we briefly introduce the syntax of `PRISM`. The basic components in `PRISM` are `modules`. A module consists of two parts: `variables` and `commands`. Variables describe possible states of a module, while commands describe its state transitions. A command is composed of a `guard` and some `updates`. For example, "$x = 0 \rightarrow 0.8 : (x' = 0) + 0.2 : (x' = 1);$" is a command with a guard "$x = 0$" and an update rule "$0.8 : (x' = 0) + 0.2 : (x' = 1)$". When the guard is met, i.e., $x = 0$ at the current time slot, x will remain at value 0 with probability 0.8 or change to 1 with probability 0.2 at the next time slot.

If the guards of multiple commands are met simultaneously at the same time slot, `PRISM` will choose exactly one of them to execute uniformly at random. However, the vertices of an SPBN $G = (V, E)$ operate concurrently. That is, at one time slot, multiple vertices have to be executed, provided that every fanin of those vertices is

DOI

evaluated. Without proper control, PRISM will randomly choose one of these vertices to execute, and may bias the probabilities specified by the SPBN. To prevent such bias, we introduce a fresh variable for each vertex to enforce an topological execution order of logic gates in the SPBN. This construction eliminates the possibility of the multi-vertex execution at one time slot, and thus preserving the probability specified in the update rules. Since G is a combinational design, this enforcement of ordering will not affect the system behavior.

Given an SPBN $G = (V, E)$ with a parameter assignment π to V_I, the procedure to compute the signal probability of an arbitrary vertex with PRISM is as follows.

1. Sort V into a topological order.

2. For each vertex $v \in V$, create a module with two Boolean variables: x_v represents the output variable of vertex v, and y_v enforces the execution order in the DTMC. Both variables are initialized to 0. Let u be the vertex preceding v in the topological order.

3. If $v \in V_I$, add a command: "$(y_u = 1 \ \& \ y_v = 0) \rightarrow \pi(v) : (x'_v = 1 \ \& \ y'_v = 1) + 1 - \pi(v) : (x'_v = 0 \ \& \ y'_v = 1)$" to the module of v.

4. If $v \in V \setminus V_I$, add command: "$(y_u = 1 \ \& \ y_v = 0) \rightarrow p_v : (x'_v = \neg f_v \ \& \ y'_v = 1) + 1 - p_v : (x'_v = f_v \ \& \ y'_v = 1)$" to the module of v.

5. Compute the signal probability of any $v \in V$ by specifying a PCTL formula $P_{=?}(F(x_v = 1))$.

DOI

The added commands describe the probabilistic behavior of a vertex. Observe that in both commands, a vertex v is executed only after its preceding vertex u is executed, and this order in enforced by the variable y_v. After the execution of v, y_v is set to 1 to trigger its successive executions. The PCTL formula $P_{=?}(F(x_v = 1))$ computes the probability of $x_v = 1$ in the future. Since in our DTMC, each node is executed only once, the PCTL formula computes the signal probability of v.

We remark that this transformation brutally encodes each gate into a module in the DTMC and suffers from the state-space explosion problem. It remains future investigation to search for better encoding.

4.4 Discussion

We discuss potential applications, extensions, and connections of probabilistic property evaluation in the following.

4.4.1 Probabilistic equivalence checking

Given two PBNs, their equivalence checking can be easily formulated under PPE and MPPE framework. The property network corresponds to a miter circuit that asserts the equivalence of corresponding outputs of the two PBNs, same as in the equivalence checking of deterministic designs. With the proposed framework, we

DOI

can analyze the average (resp. maximum) probability that the two PBNs are functionally different through PPE (resp. MPPE). We refer to the equivalence checking problem as *probabilistic equivalence checking* (PEC) for the average-case analysis and *maximum probabilistic equivalence checking* (MPEC) for the worst-case analysis. Since equivalence checking is widely encountered, our experiments will focus on PEC and MPEC to compare the strengths and weaknesses of different solutions that we proposed in Section 4.3.

4.4.2 Prioritized output requirement

For some applications, we may want to impose different criticality requirements on different output signals. Given an SPBN G over primary inputs X, internal vertices Y, and auxiliary inputs Z, this output-prioritized version of MPPE is naturally expressible in terms of stochastic integer linear programming (SILP) [99] as follows:

$$\max_{X} \ \mathbb{E}[\sum_{i=1}^{n} w_i o_i(X, Y, Z)] \ \ s.t. \ \ \phi,$$

where $o_i \in V_O$ is an output of G, $|V_O| = n$, w_i is the weight of o_i, $\mathbb{E}[\cdot]$ denotes the expectation value, and ϕ is a set of linear inequalities derived from the CNF formula of G through the standard translation from clauses to linear constraints. To illustrate, a clause $(x \vee \neg y \vee z)$ in a CNF formula is transformed into a linear inequality $(x + 1 - y + z) \geq 1$, or $(x - y + z) \geq 0$ in ϕ. Note that the worst case formulation in Theorem 4.2 is a special case of the SILP formulation where the expectation value of the miter's output is maximized.

DOI

4.4.3 Connection to approximate design analysis

Approximate design analysis assesses the deviation between an approximate design and its exact counterpart in two scenarios: the worse and average cases. For the worst-case analysis, integer linear programming (ILP) can be applied to find an input assignment to maximize the number of deviating outputs. For the average-case analysis, model counting can be used to compute the number of input assignments that make the two designs have different output responses. In both cases, our PPE framework can be applied to analyze an approximate design, which can be seen as a probabilistic design without random behavior. For probabilistic design analysis, if all random variables become deterministic, then it degenerates to approximate design analysis (from SILP to ILP under the worst-case analysis and from weighted model counting to model counting under the average-case analysis).

4.5 Evaluation

We evaluated various techniques to solve the proposed PPE formulation. Specifically, we focus on PEC and MPEC problems, as equivalence checking is widely encountered in automated design verification.

For the worst-case analysis (i.e., MPEC), two SSAT-based techniques were evaluated. The signal-probability approach via BDD-based SSAT solving is implemented in the `C` language inside the `ABC` [11] environment. The BDD operations are han-

DOI

dled by the CUDD [101] package. Our prototyping implementation[1] of the signal-probability approach is named BDDsp. A version of BDDsp without variable reordering during the BDD construction is called BDDsp-nr in the experiments. We used commit 2ff8e74 of branch master in the experiments. For the CNF-based SSAT approach, we employ the state-of-the-art DPLL-based SSAT solver DC-SSAT [67].

For the average-case analysis (i.e., PEC), in addition to the above two SSAT-based solutions, two techniques based on model counting were also evaluated. A classic exact model counter Cachet [95, 96] is used for the solution via exact weighted model counting. Thanks to the proposed formula rewriting that converts an instance of weighted model counting into an unweighted one, we are able to leverage the latest developments of approximate model counting. A state-of-the-art approximate model counter ApproxMC-4.0.1 [15, 16] is applied to solve the converted instances. To achieve a relatively fair comparison with the other exact techniques, we set the epsilon parameter to 0.99 and the delta parameter to 0.01 for ApproxMC.

4.5.1 Benchmark set

Combinational circuits from the ISCAS '85 [12] and EPFL [1] benchmark suites were used in the experiments. The circuit statistics are shown in Tables 4.1 and 4.2. We represent the circuits as and-inverter graphs (AIGs). Although the AIG represen-

[1] Available at: https://github.com/NTU-ALComLab/ssatABC

DOI

Table 4.1: Circuit statistics of ISCAS benchmark suite

Circuit	#PI	#PO	#And	#Level
c1355	41	32	504	26
c1908	33	25	414	32
c2670	233	140	717	21
c3540	50	22	1 038	41
c432	36	7	209	42
c499	41	32	400	20
c5315	178	123	1 773	38
c6288	32	32	2 337	120
c7552	207	108	2 074	29
c880	60	26	327	24

tation is chosen in our experiments, the proposed formulation and solutions are applicable to other circuit representations, such as FPGA LUT-based or standard cell-based designs.

In order to generate circuits with probabilistic errors, we specify an error rate ϵ of a (data-independent) probabilistic gate and a defection rate δ of an entire design (i.e., the ratio of erroneous gates to all gates in a design). We fixed both parameters to constants for evaluation and selected probabilistic gates at random according to the defection rate. In our experiments, ϵ was set to 0.125, and δ was set to 0.01 and 0.1. To solve probabilistic equivalence checking, a miter circuit that compares a design with probabilistic errors against its error-free counterpart is built. Since the two designs are structurally identical except for the probabilistic gates, logic synthesis may achieve substantial reduction when the number of erroneous nodes

DOI

Table 4.2: Circuit statistics of EPFL benchmark suite

Circuit	#PI	#PO	#And	#Level
adder	256	129	1 020	255
arbiter	256	129	11 839	87
bar	135	128	3 336	12
cavlc	10	11	693	16
ctrl	7	26	174	10
dec	8	256	304	3
div	128	128	57 247	4 372
hyp	256	128	214 335	24 801
i2c	147	142	1 342	20
int2float	11	7	260	16
log2	32	32	32 060	444
max	512	130	2 865	287
mem_ctrl	1 204	1 231	46 836	114
multiplier	128	128	27 062	274
priority	128	8	978	250
router	60	30	257	54
sin	24	25	5 416	225
sqrt	128	64	24 618	5 058
square	64	128	18 484	250
voter	1 001	1	13 758	70

DOI

Table 4.3: Miter statistics of ISCAS benchmark suite ($\delta = 0.01$)

Circuit	#PI	#AI	#And	#Level
c1355	41	8	488	26
c1908	33	4	424	28
c2670	233	6	443	17
c3540	50	8	1 261	38
c432	36	2	274	30
c499	41	4	484	24
c5315	178	17	1 262	31
c6288	32	20	3 086	103
c7552	207	27	1 915	30
c880	60	3	220	27

is small. In our evaluation, we synthesized a miter with the script `resyn2` in `ABC` before applying the discussed techniques. The statistics of the synthesized miters are shown in Tables 4.3, 4.4, 4.5, and 4.6. Note that the numbers of auxiliary inputs (AIs) in these tables are the numbers of erroneous gates in the circuits. Therefore, a circuit would have more AIs if the specified defection rate is larger.

4.5.2 Experimental setup

Our experiments were performed on a machine with one 2.2 GHz CPU (Intel Xeon Silver 4210) with 40 processing units and 134616 MB of RAM. The operating system was Ubuntu 20.04 (64 bit), using Linux 5.4. The programs were compiled with `g++ 9.3.0`. Each PEC or MPEC task was limited to a CPU core, a CPU time of

DOI

Table 4.4: Miter statistics of ISCAS benchmark suite ($\delta = 0.1$)

Circuit	#PI	#AI	#And	#Level
c1355	41	52	783	35
c1908	33	50	785	40
c2670	233	79	1 205	27
c3540	50	108	1 811	53
c432	36	20	390	40
c499	41	40	722	29
c5315	178	159	2 731	44
c6288	32	214	4 250	131
c7552	207	211	3 232	40
c880	60	33	615	33

15 min, and a memory usage of 15 GB. To achieve reliable benchmarking, we used a benchmarking framework `BenchExec`[2] [9].

4.5.3 Results

PEC instances

The solving results of the PEC instances with the defection rate δ equal 0.01 and 0.1 are shown in Tables 4.7 and 4.8, respectively. The average probabilities for the two circuits to have different outputs and the CPU time are reported. The symbol "-" in an entry of the tables indicates the execution ran out of computational resource.

[2] Available at: https://github.com/sosy-lab/benchexec

DOI

Table 4.5: Miter statistics of EPFL benchmark suite ($\delta = 0.01$)

Circuit	#PI	#AI	#And	#Level
adder	256	12	1 245	235
arbiter	256	128	21 590	99
bar	135	45	4 467	23
cavlc	10	13	791	21
ctrl	7	3	31	10
dec	8	8	7	3
div	128	567	83 886	4 446
hyp	256	2 103	343 274	25 380
i2c	147	17	280	23
int2float	11	3	100	12
log2	32	347	53 880	390
max	512	20	5 249	216
mem_ctrl	1 204	428	69 983	119
multiplier	128	257	40 136	271
priority	128	8	966	216
router	60	6	324	28
sin	24	66	8 938	196
sqrt	128	274	39 555	5 174
square	64	193	24 505	240
voter	1 001	132	14 167	66

DOI

Table 4.6: Miter statistics of EPFL benchmark suite ($\delta = 0.1$)

Circuit	#PI	#AI	#And	#Level
adder	256	112	1 899	294
arbiter	256	1 195	26 444	124
bar	135	335	6 838	29
cavlc	10	67	1 135	25
ctrl	7	21	144	15
dec	8	33	560	15
div	128	5 776	102 085	5 274
hyp	256	21 652	446 257	30 608
i2c	147	124	1 843	25
int2float	11	30	441	22
log2	32	3 234	68 468	499
max	512	261	6 210	280
mem_ctrl	1 204	4 700	99 030	158
multiplier	128	2 707	53 421	338
priority	128	89	1 702	294
router	60	29	441	44
sin	24	526	11 799	247
sqrt	128	2 457	45 624	6 504
square	64	1 860	35 015	302
voter	1 001	1 359	21 680	103

DOI

An instance that cannot be solved by any technique is not shown in the tables.

We observe the following phenomenons from these tables. First of all, the defection rate strongly influences the performance of all methods. As discussed above, the number of AIs is proportional to the defection rate. More AIs result in more inputs to a miter and more variables in a formula. Therefore, all of the proposed techniques performed worse when $\delta = 0.1$. Second, `BDDsp` solved many instances in the shortest time among all compared methods. Recall that the complexity of the proposed BDD-based SSAT solving is linear to the numbers of BDD nodes. The satisfying probability is readily available as long as the BDD of a miter can be built. On the other hand, `DC-SSAT` and `Cachet` are less competitive in the evaluation. Both of them failed to handle circuits with more than a thousand gates. Finally, `ApproxMC` achieved the best scalability. Although it tends to spend more time for small- and medium-sized circuits than `BDDsp` did, it uniquely solved three instances when $\delta = 0.01$ and four instances when $\delta = 0.1$. Enabling any model counter to deal with weighted model counting, the proposed formula-rewriting technique demonstrates its value in improving the scalability to solve the PEC problem.

MPEC instances

The solving results of the MPEC instances with the defection rate δ equal 0.01 and 0.1 are shown in Tables 4.9 and 4.10, respectively. The maximum probabilities for the two circuits to have different outputs and the CPU time are reported. The

DOI

Table 4.7: Solving PEC by various techniques ($\delta = 0.01$)

	BDDsp		DC-SSAT		Cachet		ApproxMC	
Circuit	T (s)	Pr	T (s)	Pr	T (s)	Pr	T (s)	Pr
adder	0.37	$7.28e{-1}$	–	–	–	–	669.68	$7.19e{-1}$
bar	7.04	$9.85e{-1}$	–	–	–	–	583.60	$1.00e{+0}$
c1355	5.20	$4.32e{-1}$	–	–	–	–	29.54	$4.30e{-1}$
c1908	0.49	$6.25e{-2}$	–	–	–	–	13.48	$6.05e{-2}$
c2670	0.27	$3.10e{-1}$	–	–	–	–	474.74	$3.13e{-1}$
c3540	8.26	$2.28e{-1}$	–	–	–	–	43.87	$2.30e{-1}$
c432	0.06	$3.15e{-2}$	–	–	0.29	$3.15e{-2}$	11.66	$3.13e{-2}$
c499	2.14	$2.62e{-1}$	–	–	–	–	19.82	$2.66e{-1}$
c5315	65.66	$6.53e{-1}$	–	–	–	–	451.96	$6.56e{-1}$
c6288	–	–	–	–	–	–	69.32	$9.06e{-1}$
c7552	–	–	–	–	–	–	654.42	$7.03e{-1}$
c880	0.63	$1.23e{-1}$	–	–	11.01	$1.23e{-1}$	31.66	$1.25e{-1}$
cavlc	0.05	$4.96e{-2}$	0.15	$4.96e{-2}$	0.11	$4.96e{-2}$	17.60	$4.98e{-2}$
ctrl	0.04	$1.87e{-1}$	0.01	$1.87e{-1}$	0.04	$1.87e{-1}$	1.05	$1.88e{-1}$
dec	0.04	$6.56e{-1}$	0.01	$6.56e{-1}$	0.04	$6.56e{-1}$	6.03	$6.56e{-1}$
i2c	0.08	$4.33e{-1}$	–	–	742.00	$4.33e{-1}$	311.88	$4.22e{-1}$
int2float	0.04	$6.39e{-3}$	0.01	$6.39e{-3}$	0.04	$6.39e{-3}$	1.04	$6.47e{-3}$
priority	0.12	$3.93e{-1}$	–	–	–	–	170.12	$3.91e{-1}$
router	0.05	$9.40e{-4}$	–	–	6.84	$9.40e{-4}$	36.86	$9.16e{-4}$
sin	–	–	–	–	–	–	490.12	$1.00e{+0}$

DOI

Table 4.8: Solving PEC by various techniques ($\delta = 0.1$)

	BDDsp		DC-SSAT		Cachet		ApproxMC	
Circuit	T (s)	Pr	T (s)	Pr	T (s)	Pr	T (s)	Pr
adder	1.77	$1.00e{+}0$	–	–	–	–	–	–
c1355	–	–	–	–	–	–	292.21	$9.22e{-}1$
c1908	–	–	–	–	–	–	248.79	$9.22e{-}1$
c432	11.19	$4.99e{-}1$	–	–	–	–	71.26	$4.84e{-}1$
c499	–	–	–	–	–	–	189.28	$8.13e{-}1$
c880	–	–	–	–	–	–	185.38	$8.75e{-}1$
cavlc	0.26	$6.89e{-}1$	–	–	1.77	$6.89e{-}1$	331.78	$6.72e{-}1$
ctrl	0.05	$8.22e{-}1$	0.05	$8.22e{-}1$	0.05	$8.22e{-}1$	33.26	$8.28e{-}1$
dec	0.05	$9.87e{-}1$	184.36	$9.87e{-}1$	1.34	$9.87e{-}1$	81.69	$1.00e{+}0$
int2float	0.06	$4.32e{-}1$	4.30	$4.32e{-}1$	0.36	$4.32e{-}1$	72.68	$4.30e{-}1$
router	0.36	$1.76e{-}1$	–	–	–	–	156.31	$1.76e{-}1$

symbol "-" in an entry of the tables indicates the execution ran out of computational resource. An instance that cannot be solved by any technique is not shown in the tables.

We included `BDDsp-nr` (i.e., `BDDsp` without variable reordering) in the MPEC experiments. In order to achieve a smaller BDD, the `CUDD` package automatically reorders variables during the BDD construction by default. Variable reordering usually achieves better results in our empirical experience. For PEC instance, since both the PI and the AI variables are randomly quantified, the variable order in BDD construction is irrelevant to signal probability computation. Therefore, we always enabled this option in the PEC experiments.

DOI

Table 4.9: Solving MPEC by various techniques ($\delta = 0.01$)

	BDDsp		BDDsp-nr		DC-SSAT	
Circuit	T (s)	Pr	T (s)	Pr	T (s)	Pr
adder	3.65	$7.99e{-1}$	–	–	–	–
c1355	22.33	$6.56e{-1}$	1.58	$6.56e{-1}$	–	–
c1908	0.92	$4.14e{-1}$	0.23	$4.14e{-1}$	47.78	$4.14e{-1}$
c2670	0.32	$5.51e{-1}$	–	–	–	–
c3540	28.16	$6.07e{-1}$	19.24	$6.07e{-1}$	–	–
c432	0.06	$2.34e{-1}$	0.05	$2.34e{-1}$	–	–
c499	3.32	$4.14e{-1}$	0.39	$4.14e{-1}$	–	–
c880	0.67	$3.30e{-1}$	0.99	$3.30e{-1}$	–	–
cavlc	1.51	$5.42e{-1}$	0.05	$5.42e{-1}$	0.15	$5.42e{-1}$
ctrl	0.05	$2.34e{-1}$	0.04	$2.34e{-1}$	0.01	$2.34e{-1}$
dec	0.05	$6.56e{-1}$	0.05	$6.56e{-1}$	0.01	$6.56e{-1}$
i2c	–	–	0.37	$8.57e{-1}$	–	–
int2float	0.05	$2.34e{-1}$	0.04	$2.34e{-1}$	0.01	$2.34e{-1}$
priority	1.78	$6.34e{-1}$	0.07	$6.34e{-1}$	–	–
router	0.05	$5.42e{-1}$	0.05	$5.42e{-1}$	–	–

However, variable reordering is problematic under the MPEC scenario. For MPEC instances, since the PI variables are existentially quantified and the AI variables are randomly quantified, the PI variables must be placed before the AI variables. The default reordering, nevertheless, might violate such structure. To make variable reordering respect the quantification prefix, we classify PI and AI variables into two separate groups and require that variable reordering can only happen within each group. In the following experiments, we use `BDDsp-nr` as a baseline to evaluate the performance of the constrained variable reordering.

DOI

Table 4.10: Solving MPEC by various techniques ($\delta = 0.1$)

	BDDsp		BDDsp-nr		DC-SSAT	
CIRCUIT	T (s)	Pr	T (s)	Pr	T (s)	Pr
cavlc	–	–	0.06	$9.78e{-1}$	–	–
ctrl	0.05	$8.65e{-1}$	0.05	$8.65e{-1}$	0.06	$8.65e{-1}$
dec	0.06	$9.88e{-1}$	0.05	$9.88e{-1}$	181.13	$9.88e{-1}$
int2float	–	–	0.05	$9.01e{-1}$	4.21	$9.01e{-1}$
router	–	–	2.75	$8.96e{-1}$	–	–

From these tables, we observe the following phenomenons. First, the property violation probabilities between the average case and the worst case can differ for several orders of magnitude. For example, the difference for circuit `router` when $\delta = 0.01$ is about three orders of magnitude. This observation shows the criticality of the MPPE formulation for scenarios where the worst-case analysis is concerned. Second, same as the PEC experiments, `BDDsp` performed much better than `DC-SSAT` did. Regarding the reordering of variables, the performance difference is not very consistent. There are cases which can be solved by `BDDsp` within seconds but cannot be solved by `BDDsp-nr`, and vice versa. Constrained variable reordering is still useful in general. However, if the original variable order happens to be good enough for BDD construction, enabling the constrained reordering would just incur overhead for CPU time.

The above results on the PEC and MPEC instances suggest that:

- The proposed BDD-based SSAT solver `BDDsp` achieves the best performance

DOI

for small- and medium-sized circuits, whose BDDs can be constructed easily.

- The state-of-the-art approximate model counter `ApproxMC` has the best scalability in our evaluation, which shows the unique value of the proposed formula rewriting.

- The exact weighted model counter `Cachet` and the CNF-based SSAT solver `DC-SSAT` do not scale well when the circuit size grows.

DOI

Chapter 5

Random-Exist Quantified SSAT

In this chapter, we propose a new algorithm that combines modern SAT-solving and model-counting techniques for random-exist quantified SSAT formulas. Most content in this chapter is based on our conference paper [56] published at IJCAI'17.

5.1 Preliminaries

A random-exist quantified SSAT formula Φ has the form $\text{ƨ}X, \exists Y.\phi(X, Y)$, where X and Y are two disjoint sets of Boolean variables, and $\phi(X, Y)$ is a CNF formula.

DOI

5.1.1 Generalization of SAT/UNSAT minterms

Given an assignment τ over X, if $\phi(X,Y)|_\tau$ is satisfiable (resp. unsatisfiable), τ is called a SAT (resp. an UNSAT) minterm of ϕ over X. The generalization process of a SAT or an UNSAT minterm τ aims at expanding it to a cube τ^+, while maintaining the satisfiability of $\phi(X,Y)|_{\tau^+}$ the same as $\phi(X,Y)|_\tau$.

Example 5.1. *Consider formula $\phi(x_1,x_2,y_1,y_2) = x_1 \wedge (\neg x_2 \vee y_1 \vee y_2)$. The complete assignment $\tau = x_1x_2$ over X, i.e., $\tau(x_1) = \top, \tau(x_2) = \top$, is a SAT minterm of ϕ over X because $\phi|_\tau$ is satisfiable by the assignment $\mu = y_1y_2$. On the other hand, the partial assignment $\tau^+ = \neg x_1$, i.e., $\tau^+(x_1) = \bot$, is an UNSAT cube of ϕ as $\phi|_{\tau^+}$ is unsatisfiable.*

Minimum Satisfying Assignment

For a CNF formula $\phi(X,Y)$, let τ be a SAT minterm over X and let μ be a satisfying complete assignment for the induced formula $\phi(X,Y)|_\tau$ over Y. To generalize τ into a cube, one can find a subset of literals from τ and μ that are able to satisfy all clauses in ϕ while the number of literals taken from τ is as few as possible. If some literals in τ are irrelevant to the satisfiability, they can be dropped from τ, thus expanding τ to a SAT cube τ^+. The generalized cube τ^+ is called a *minimum satisfying assignment* if the number of literals taken from τ is minimized. The process of finding a minimum satisfying assignment is also known as finding a *minimum hitting set*.

DOI

Minimum Conflicting Assignment

Given an UNSAT minterm τ of a formula ϕ, modern SAT solvers, e.g., `MiniSat` [31, 32], are able to compute a conjunction of literals from τ that is responsible for the conflict. If some literals in τ are irrelevant to the conflict, they are dropped from τ, thus expanding τ to an UNSAT cube τ^+. If the number of literals in an UNSAT cube τ^+ is minimized, τ^+ is called a *minimum conflicting assignment*. The process of finding a minimum conflicting assignment is also known as finding a *minimum UNSAT core*.

5.2 Solving random-exist quantified SSAT

Consider a random-exist quantified SSAT formula $\Phi = \text{ƀ}X, \exists Y.\phi(X, Y)$. The satisfying probability of Φ equals the summation of weight of all SAT minterms over X, or, equivalently, 1 minus the summation of weight of all UNSAT minterms over X. To identify an assignment τ over X as a SAT or an UNSAT minterm, it suffices to check whether $\phi(X, Y)|_\tau$ is satisfiable or not. A naive solution to computing the satisfying probability of Φ is to exhaustively examine all assignments over X, classify them as SAT or UNSAT minterms, and aggregate the weight of collected minterms.

The above naive idea can be improved by exploiting the minterm-generalization techniques discussed in Section 5.1.1. For instance, in Example 5.1, $\tau = x_1x_2$ is

DOI

a SAT minterm over $\{x_1, x_2\}$ for $\phi(x_1, x_2, y_1, y_2) = x_1 \wedge (\neg x_2 \vee y_1 \vee y_2)$. Observe that $\phi(x_1, x_2, y_1, y_2)$ is satisfiable under the partial assignment $\tau^+ = x_1$. In other words, the SAT minterm τ can be generalized into the SAT cube τ^+, which contains two minterms. Through the generalization analysis, multiple minterms can be collected in a single SAT-solving run, enhancing the efficiency to enumerate all possible assignments over X. As will be shown in Section 5.4, the minterm-generalization techniques are essential to the efficiency of the proposed algorithm. However, the weight of each collected cube cannot be summed up directly due to the potential overlapping between generalized cubes. This difficulty will be overcome by applying weighted model counting to evaluate the weight of the collected cubes.

The above thoughts give rise to the proposed algorithm in Alg. 4 to compute the satisfying probability of $\Phi = \mathsf{Ƨ}X, \exists Y.\phi(X, Y)$. The proposed algorithm works as follows. Fow now, assume the run-time limit `TO` to be infinity. The effect of imposing a run-time limit on Alg. 4 will be explained in Section 5.2.4. Two SAT solvers are used in Alg. 4. In addition to the SAT solver that holds the matrix CNF $\phi(X, Y)$, the other SAT solver $\psi(X)$, named the selector in the following, is initialized as a tautology, i.e., without clauses. The selector $\psi(X)$ is in charge of selecting an assignment τ over X. After τ is chosen, the matrix solver $\phi(X, Y)$ will check whether $\phi(X, Y)|_\tau$ is satisfiable or not. If $\phi(X, Y)|_\tau$ is satisfiable, τ will be generalized into a SAT cube by the subroutine `MinimalSatisfying`; if $\phi(X, Y)|_\tau$ is unsatisfiable, τ will be generalized into an UNSAT cube by the subroutine `MinimalConflicting`.

DOI

Algorithm 4 Solving random-exist quantified SSAT formulas

Input: $\Phi = \text{Ƨ}X, \exists Y.\phi(X, Y)$ and a run-time limit `TO`

Output: Lower and upper bounds (P_L, P_U) of $\Pr[\Phi]$

1: $\psi(X) := \top$

2: $C_\top := \emptyset$

3: $C_\bot := \emptyset$

4: **while** ($\texttt{SAT}(\psi)$ **and** `run-time` $<$ `TO`) **do**

5: $\quad \tau := \psi.\text{model}$

6: $\quad$ **if** ($\texttt{SAT}(\phi|_\tau)$) **then**

7: $\quad\quad \tau^+ := \texttt{MinimalSatisfying}(\phi, \tau)$

8: $\quad\quad C_\top := C_\top \cup \{\tau^+\}$

9: $\quad$ **else**

10: $\quad\quad \tau^+ := \texttt{MinimalConflicting}(\phi, \tau)$

11: $\quad\quad C_\bot := C_\bot \cup \{\tau^+\}$

12: $\quad$ **end if**

13: $\quad \psi := \psi \wedge \neg\tau^+$

14: **end while**

15: **return** $(\texttt{ComputeWeight}(C_\top), 1 - \texttt{ComputeWeight}(C_\bot))$

DOI

Instead of finding a *minimum* satisfying or conflicting assignment, which is computationally expensive, we resort to finding a *minimal* satisfying or conflicting assignment, i.e., an assignment that has no literals removable without affecting the (un)satisfiability, to leverage the efficient UNSAT-core computation for effective generalization. After τ is generalized to τ^+ and enlisted in $C_\perp$ or $C_\top$, the negation of τ^+, which becomes a blocking clause, will be conjoined with ψ to prune the assignments contained by τ^+.

The above process is repeated until ψ becomes unsatisfiable, which signifies the Boolean space spanned by X has been exhaustively searched. The subroutine `ComputeWeight` is then invoked to evaluate the weight of the collected cubes. The subroutines `MinimalConflicting`, `MinimalSatisfying`, and `ComputeWeight` will be detailed below.

5.2.1 Minimal satisfying assignment

Given a SAT minterm τ over X, let μ be a satisfying assignment over Y for $\phi(X,Y)|_\tau$. The subroutine `MinimalSatisfying` generalizes τ to τ^+ by the following steps.

a) Remove every clause C in $\phi(X,Y)|_\tau$ that contains some true literal from μ.

b) For each literal l in τ, drop l and examine whether the rest of clauses remain satisfied by scanning these clauses and checking if each of them still contains some true literal. If the rest of clauses are still satisfied, discard l; otherwise,

DOI

put l in τ^+.

After the above steps, the SAT minterm τ will be generalized into a minimal satisfying assignment τ^+.

5.2.2 Minimal conflicting assignment

Let τ be an UNSAT minterm over X for $\phi(X, Y)$. The analysis of unsatisfiability can be done with a modern SAT solver (e.g., using function `analyzeFinal()` in `MiniSat`) to find a conjunction of literals from τ responsible for the conflict. However, in general this conjunction of literals might not be minimal, and some of the literals could be dropped. The subroutine `MinimalConflicting` takes the conjunction of literals responsible for the conflict computed by a SAT solver and makes it minimal as follows. For each literal l in the conjunction, drop l and examine whether $\phi(X, Y)$ remains unsatisfiable by invoking a SAT call. If it is unsatisfiable, discard l; otherwise, put l in τ^+. After the above steps, the UNSAT minterm τ will be generalized into a minimal conflicting assignment τ^+.

5.2.3 Weight computation

The subroutine `ComputeWeight` aggregates the weight of collected cubes by invoking a weighted model counter. Because a weighted model counter takes CNF formulas as input, `ComputeWeight` first negates each collected cube to turn it into a clause,

DOI

and conjoins the resulting clauses into a CNF formula. As the CNF formula is the negation of the disjunction of the cubes, the weight of the cubes equals one minus the weight of the CNF formula, which is computed by a weighted model counter.

5.2.4 Modification for approximate SSAT

The proposed algorithm can be easily modified to solve *approximate SSAT*, where upper and lower bounds of the satisfying probability of an SSAT formula are computed. Suppose Alg. 4 is forced to terminate before the selector ψ becomes unsatisfiable. The weights of the collected SAT and UNSAT cubes are still valid and can be aggregated by `ComputeWeight`, and the resulted weights reflect the lower and upper bounds of the satisfying probability, respectively. The early termination can be triggered by imposing a run-time limit for Alg. 4. Compared to previous DPLL-based approaches that branch on a single variable, the proposed algorithm considers all randomly quantified variables together and exploits the concept of SAT and UNSAT cubes over the Boolean space spanned by randomly quantified variables, making the intermediate collected SAT and UNSAT cubes convey useful information about the upper and lower bounds of the exact satisfying probability. As will be seen in our experiments over formulas arising from VLSI circuit analysis, the proposed algorithm is able to find tight bounds even with a short run-time limit. Compared to the DPLL-based state-of-the-art methods, which cannot be easily modified for approximate SSAT, the proposed method enjoys the flexibility of solving SSAT approximately or

DOI

exactly, depending on the imposed run-time constraint.

We note that the proposed algorithm is more efficient in memory consumption than previous DPLL-based algorithms. Prior DPLL-based algorithms mostly apply subproblem memorization to avoid repeated computation on the same subproblem. However, without special treatment, such memorization may result in rapid growth in memory usage. On the other hand, in the proposed algorithm, the numbers of collected cubes are greatly reduced by the minterm-generalization techniques, which gives rise to the memory efficiency. In our empirical evaluation, the proposed algorithm consumed two orders of magnitude less memory than the state-of-the-art DPLL-based solver.

The following example illustrates how Alg. 4 solves a random-exist quantified SSAT formula.

Example 5.2. *Consider a random-exist quantified SSAT formula*

$$\Phi = \text{Ꙗ}^{0.5} r_1, \text{Ꙗ}^{0.5} r_2, \text{Ꙗ}^{0.5} r_3, \exists e_1, \exists e_2, \exists e_3 . \phi,$$

with ϕ consisting of the following clauses:

$C_1 : (r_1 \vee r_2 \vee e_1)$

$C_2 : (r_1 \vee \neg r_3 \vee e_2)$

$C_3 : (r_2 \vee \neg r_3 \vee \neg e_1 \vee \neg e_2)$

$C_4 : (r_3 \vee e_3)$

DOI

$C_5 : (r_3 \vee \neg e_3)$

In the beginning, the selector $\psi(r_1, r_2, r_3)$ *is initialized without clauses, and the sets* $C_\top$ *and* $C_\bot$ *to collect SAT and UNSAT cubes are empty. Suppose* ψ *first selects an assignment* $\tau_0 = \neg r_1 \neg r_2 \neg r_3$*. Since* $\phi|_{\tau_0}$ *is unsatisfiable due to the conflict between* C_4 *and* C_5*, the subroutine* `MinimalConflicting` *returns* $\tau_0^+ = \neg r_3$*, which is the minimal conflicting assignment responsible for this conflict. Note that this minimal conflicting assignment* τ_0^+ *reflects an upper bound of* 0.5 *for* $\Pr[\Phi]$*. The selector* ψ *is then strengthened through conjunction with the negation of* τ_0^+ *to block the searched subspace. Next, suppose* $\tau_1 = \neg r_1 \neg r_2 r_3$ *is selected. Under* τ_1*, formula* $\phi|_{\tau_1}$ *is unsatisfiable due to the conflict among clauses* C_1*,* C_2*, and* C_3*, and the minimal conflicting assignment* τ_1^+ *equals* $\neg r_1 \neg r_2$*. After conjoining* ψ *with* $\neg \tau_1^+$*, suppose* $\tau_2 = \neg r_1 r_2 r_3$ *is chosen. Formula* $\phi|_{\tau_2}$ *is satisfiable through the assignment* $\mu_2 = \neg e_1 e_2 \neg e_3$*. The subroutine* `MinimalSatisfying` *is invoked to generalize* τ_2 *to* $\tau_2^+ = r_2 r_3$*, which reflects a lower bound of* 0.25 *for* $\Pr[\Phi]$*. Similarly, the negation of* τ_2^+ *is conjoined with* ψ*. Next, let the assignment chosen by* ψ *be* $\tau_3 = r_1 \neg r_2 r_3$*. Since* $\phi|_{\tau_3}$ *is satisfiable through the assignment* $\mu_3 = \neg e_1 \neg e_2 \neg e_3$*, assignment* τ_3 *is generalized to* $\tau_3^+ = r_1 r_3$ *by* `MinimalSatisfying`*. After conjoined with* $\neg \tau_3^+$*, formula* ψ *becomes unsatisfiable, which indicates the Boolean space over* $\{r_1, r_2, r_3\}$ *has been explored exhaustively. At the end, we have* $C_\bot = \{\tau_0^+, \tau_1^+\} = \{\neg r_3, \neg r_1 \neg r_2\}$ *and* $C_\top = \{\tau_2^+, \tau_3^+\} = \{r_2 r_3, r_1 r_3\}$*. The subroutine* `ComputeWeight` *is finally invoked and returns* 0.375 *as the satisfying probability of* Φ*.*

DOI

For approximate SSAT solving, suppose the procedure is forced to terminate right after τ_2^+ *is collected. The subroutine* `ComputeWeight` *will be invoked over* $C_\top = \{r_2 r_3\}$ *and* $C_\perp = \{\neg r_3, \neg r_1 \neg r_2\}$*. The cubes in* $C_\top$ *or* $C_\perp$ *are negated into CNF formulas for weighted model counting. To compute an upper bound, the UNSAT cubes* $\neg r_3$ *and* $\neg r_1 \neg r_2$ *are rewritten into a CNF formula* $(r_3) \wedge (r_1 \vee r_2)$ *and yields a probability of* 0.375 *with respect to the weights specified by the prefix. This probability is the satisfying probability of the negation of the UNSAT cubes, which gives an upper bound of* 0.375 *for* $\Pr[\Phi]$*. Similarly, we can obtain a lower bound of* 0.25 *for* $\Pr[\Phi]$ *from the SAT cube* $r_2 r_3$*.*

5.3 Applications

In this section, we discuss several applications of random-exist quantified SSAT formulas.

5.3.1 Probability of success in planning

Many planning problems can be formulated in terms of forall-exist quantified QBFs, i.e., QBFs of the form $\Phi = \forall X, \exists Y. \phi(X, Y)$. Changing the universal quantifiers of these QBFs to randomized ones yields random-exist quantified SSAT formulas. Under the game interpretation of QBFs, the satisfying probability of such an SSAT formula corresponds to the likelihood for the existential player to win the game if

DOI

the universal player decides its moves at random. In Section 5.4, we will use the *strategic companies* problem [14] as an example to evaluate SSAT solvers on planning applications.

5.3.2 Probabilistic circuit verification

The second application is the formal verification of *probabilistic design*. As probabilistic errors are becoming more common in advanced nanometer technology, the *probabilistic equivalence checking* (PEC) problem asks to compute the probability for a probabilistic circuit to produce different outputs from its faultless specification. PEC can be encoded into a random-exist quantified SSAT formula [54].

5.4 Evaluation

We evaluated the proposed Alg. 4 against the state-of-the-art DPLL-based SSAT solver `DC-SSAT` [67] over three families of random-exist quantified SSAT formulas. The proposed algorithm is implemented in the `C++` language inside the `ABC` [11] environment. The SAT solver `MiniSat`-2.2 [31] and the model counter `Cachet` [95] are used as underlying computational engines. Our prototyping implementation[1] is named `reSSAT`. A bare version of `reSSAT` without minterm generalization is called

[1]Available at: https://github.com/NTU-ALComLab/ssatABC

DOI

`reSSAT-b` in the experiments. We used commit `2ff8e74` of branch `master` for the evaluation. The solver `DC-SSAT`, which is also implemented in the `C++` language, was kindly provided by its author Majercik.

5.4.1 Benchmark set

We evaluated the SSAT solvers with both random and application formulas. These formulas are hosted in a publicly available repository[2]. We used commit `ea9fbae` of branch `master` in the experiments.

Random k-CNF formulas

The random k-CNF formulas are generated using the CNF generator `CNFgen` [53]. Let k be the number of literals in a clause, n be the number of variables of a formula, and m be the number of clauses of a formula. The CNF formulas were generated with the following parameter settings. Let k range from 3 to 9, n equal 10, 20, 30, 40, and 50, and the clauses-to-variables ratio $\frac{m}{n}$ range from $k-1$ to $k+2$. For each combination of parameters, five formulas were sampled. As a result, 700 CNF formulas were generated. To convert the generated formulas to random-exist quantified SSAT formulas, half of the variables in a formula are randomly quantified with probability 0.5, and the rest of the variables are existentially quantified.

[2]Available at: https://github.com/NTU-ALComLab/ssat-benchmarks

DOI

Application formulas

There are two families of application formulas. The first family consists of formulas that encode a planning problem *Strategic-Company* [14]. We briefly describe the problem as follows. Suppose a businessman owns n companies that produce m different kinds of products. A company is *strategic* if it is in a minimal set of companies that together produce all kinds of products. The information about a company being strategic is valuable to the businessman. Suppose the businessman considers selling out some companies upon a financial crisis, but still hopes to produce every kind of products. The businessman would prefer selling out a non-strategic company. The problem becomes more complicated if the *controlling relations* are taken into account. If a company is *controlled* by some other companies, the company can be sold out only if some of its controlling companies is also sold out. The problem to decide whether a company is strategic can be encoded as a forall-exist quantified QBF [33, 58].

We modify the QBFs encoding the strategic-company problem to their SSAT variants by replacing the universal quantifiers in the original QBFs with randomized ones with probabilities 0.5. These QBFs are taken from `QBFLIB` [82]. The satisfying probability reflects the likelihood for a company to be strategic. The QBFs that we experimented with have the following parameter settings: n equals 5, 10, 15, ..., 75, $m = 3n$, and the number of controlling relations equals 4, 9, 14, and 19. In total, there are 60 formulas in the *Strategic-Company* family.

DOI

The second family consists of formulas that encode the average-case analysis of equivalence checking for probabilistic designs. These PEC formulas are borrowed from Chapter 4 and their creation is briefly explained as follows. For more details, please refer to Section 4.5. A circuit with probabilistic errors is generated by randomly assigning gates in its faultless counterpart to be erroneous. Two parameters are specified to control the generation of probabilistic circuits. The error rate ϵ controls the probability of the occurrence of an error at a logic gate. The defect rate δ controls the ratio of the number of erroneous gates to the total number of gates in a design. These SSAT formulas are derived from the ISCAS '85 [12] and EPFL [1] benchmark suites with $\epsilon = 0.125$ and $\delta = 0.01, 0.1$. In total, there are 60 formulas in the *PEC* family.

5.4.2 Experimental setup

Our experiments were performed on a machine with one 2.2 GHz CPU (Intel Xeon Silver 4210) with 40 processing units and 134 616 MB of RAM. The operating system was Ubuntu 20.04 (64 bit), running Linux 5.4. The programs were compiled with `g++ 9.3.0`. Each SSAT-solving task was limited to a CPU core, a CPU time of 15 min, and a memory usage of 15 GB. To achieve reliable benchmarking, we used a benchmarking framework `BenchExec`[3] [9].

[3] Available at: https://github.com/sosy-lab/benchexec

DOI

5.4.3 Results

Random k-CNF formulas

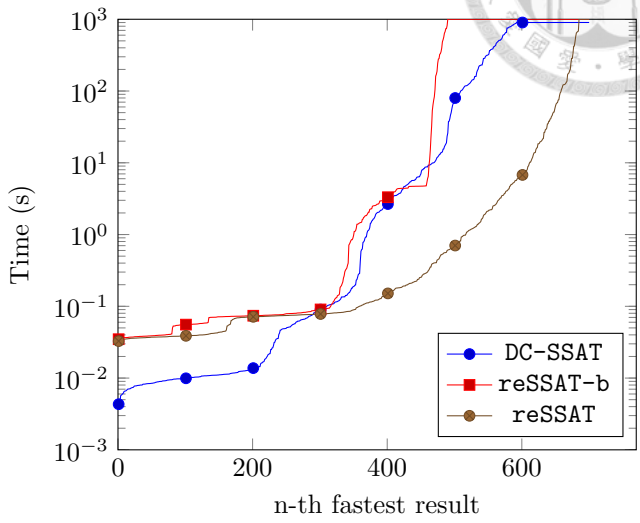
Fig. 5.1 shows the quantile plots regarding CPU time and memory usage of the SSAT instances derived from the random k-CNF formulas. A data point (x, y) in a quantile plot indicates that there are x formulas processed by the respective algorithm within a resource constraint of y. In Fig. 5.1a, we observe that `reSSAT` not only solved more formulas than `DC-SSAT`, but was also more efficient. Moreover, the minterm-generalization technique is crucial for the performance of `reSSAT`, as can be seen from the huge gap between `reSSAT` and `reSSAT-b`. On the other hand, Fig. 5.1b shows that the memory usage of `DC-SSAT` is about two orders of magnitude greater than that of `reSSAT` for large formulas. This can be attributed to the subformula caching of `DC-SSAT`. Instead, `reSSAT` stores information as SAT or UNSAT cubes, which are more compact than subformulas.

Application formulas

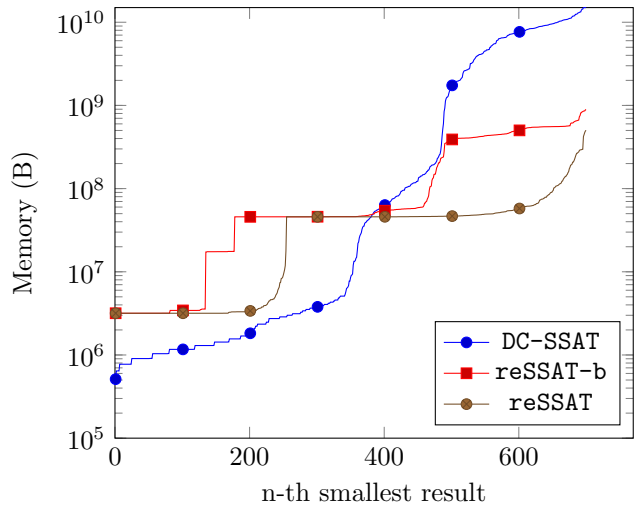
The solving results for the *Strategic-Company* family are summarized in Table 5.1. Observe that `reSSAT` successfully solved all formulas in this family, while `DC-SSAT` and `reSSAT-b` only solved 28 and 12 formulas, respectively. The results show that `reSSAT` also works well over structured instances from AI applications.

Fig. 5.2 shows the quantile plots for the *Strategic-Company* family. The quantile

DOI



(a) CPU time



(b) Memory usage

Figure 5.1: Quantile plots of random k-CNF formulas

DOI

Table 5.1: Summary of the results for 60 strategic-company formulas

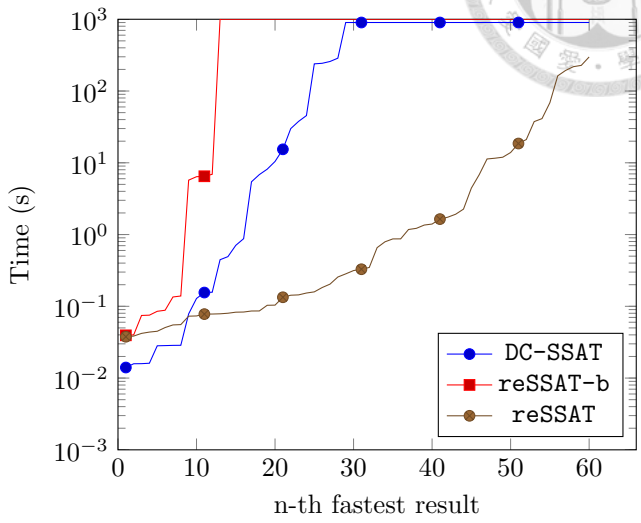
Algorithm	DC-SSAT	reSSAT	reSSAT-b
Solved formulas	28	60	12
Timeouts	32	0	48
Out of memory	0	0	0
Other inconclusive	0	0	0

plots show that reSSAT is not only effective but also efficient in terms of CPU time and memory usage. The performance gap between reSSAT and reSSAT-b again confirms the importance of the minterm-generalization technique. Fig. 5.3 shows the scatter plots, which indicate the great improvement for the run-time efficiency of reSSAT.

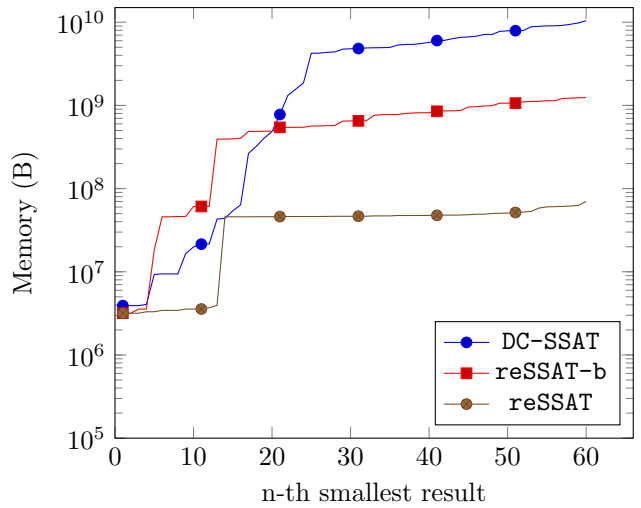
The solving results for the *PEC* family are summarized in Table 5.2. Note that all the evaluated SSAT solvers failed to exactly solve most formulas in this family. This phenomenon is similar to a concluding remark from Chapter 4 that states CNF-based SSAT solvers do not scale nicely when the circuit size grows. Nevertheless, recall that the proposed Alg. 4 is able to derive upper and lower bounds of the satisfying probability even if it could not exactly solve a large instance. We will investigate the approximation ability of reSSAT in the following.

Tables 5.3 and 5.4 show the results of solving PEC formulas with δ equal to 0.01 and 0.1, respectively. As BenchExec sends a timeout signal to reSSAT after 15 min,

DOI



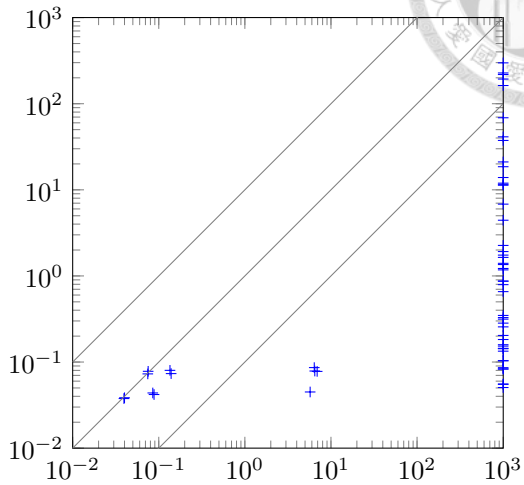
(a) CPU time



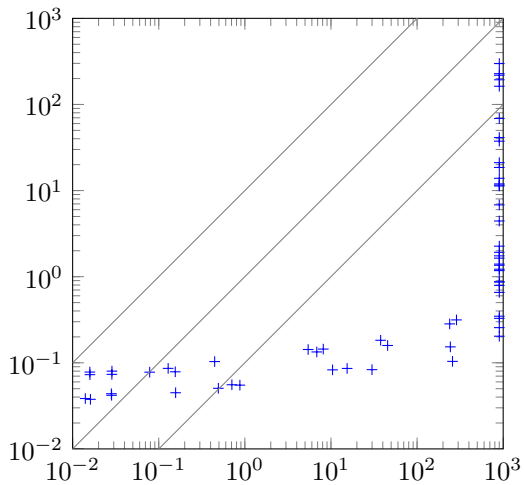
(b) Memory usage

Figure 5.2: Quantile plots of strategic-company formulas

DOI



(a) `reSSAT-b`



(b) `DC-SSAT`

Figure 5.3: Run-time scatter plots of strategic-company formulas with `reSSAT` in y-axis and compared approaches in x-axis

DOI

Table 5.2: Summary of the results for 60 PEC formulas

Algorithm	DC-SSAT	reSSAT	reSSAT-b
Solved formulas	7	3	3
Timeouts	15	57	57
Out of memory	28	0	0
Other inconclusive	10	0	0

we allow an additional 100 sec for reSSAT to compute the weights of the collected cubes. In our evaluation, the numbers of the SAT cubes are often much greater than those of the UNSAT cubes. In order to successfully deliver useful information upon timeout, reSSAT will first invoke Cachet over the UNSAT cubes to calculate an upper bound. After Cachet finishes this weight-computation query, reSSAT will call it again over the SAT cubes to obtain a lower bound. Unfortunately, due to the huge numbers of SAT cubes, Cachet failed to complete the queries within the additional 100 sec. As a result, the following tables only report upper bounds (UB) when reSSAT suffered from timeouts. Nevertheless, this is only a technical limitation in our evaluation, and it should not be misunderstood as the proposed Alg. 4 cannot derive lower bounds.

From these tables, we observe the following phenomenons. First, reSSAT was able to exactly solve a formula or derive a very tight upper bound when DC-SSAT solved the formula exactly. Second, reSSAT was able to compute useful upper bounds

DOI

Table 5.3: Results of solving the PEC formulas ($\delta = 0.01$)

	DC-SSAT		reSSAT			reSSAT-b		
FORMULA	T (s)	Pr	T (s)	Pr	UB	T (s)	Pr	UB
adder	–	–	–	–	$7.98e{-1}$	–	–	–
bar	–	–	–	–	$9.98e{-1}$	–	–	–
c1908	–	–	–	–	–	–	–	$2.65e{-1}$
c2670	–	–	–	–	$5.03e{-1}$	–	–	$1.00e{+0}$
c3540	–	–	–	–	$2.98e{-1}$	–	–	$9.96e{-1}$
c432	–	–	–	–	$3.15e{-2}$	–	–	$5.92e{-1}$
c5315	–	–	–	–	$9.66e{-1}$	–	–	–
c6288	–	–	–	–	$1.00e{+0}$	–	–	$1.00e{+0}$
c880	–	–	–	–	$1.74e{-1}$	–	–	$8.83e{-1}$
cavlc	0.16	$4.96e{-2}$	–	–	$4.96e{-2}$	–	–	$4.96e{-2}$
ctrl	0.01	$1.87e{-1}$	0.08	$1.87e{-1}$	–	0.08	$1.87e{-1}$	–
dec	0.01	$6.56e{-1}$	0.07	$6.56e{-1}$	–	10.04	$6.56e{-1}$	–
i2c	–	–	–	–	$8.18e{-1}$	–	–	$8.07e{-1}$
int2float	0.02	$6.39e{-3}$	0.08	$6.39e{-3}$	–	0.11	$6.39e{-3}$	–
max	–	–	–	–	$9.74e{-1}$	–	–	–
priority	–	–	–	–	$7.61e{-1}$	–	–	–
router	–	–	–	–	$1.41e{-3}$	–	–	$2.41e{-1}$

on larger formulas which DC-SSAT failed to solve. Compared to Tables 4.7 and 4.8, these upper bounds are often close to the exact probabilities obtained by the BDD-based SSAT solver. Third, the proposed minterm-generalization technique helps to tighten the obtained upper bounds.

The above results on the random and application formulas suggest that:

- The proposed solver reSSAT outperforms DC-SSAT in terms of both CPU time

DOI

Table 5.4: Results of solving the PEC formulas ($\delta = 0.1$)

	DC-SSAT		reSSAT			reSSAT-b		
Formula	T (s)	Pr	T (s)	Pr	UB	T (s)	Pr	UB
c1908	–	–	–	–	$1.00e{+}0$	–	–	–
c3540	–	–	–	–	$1.00e{+}0$	–	–	–
c432	–	–	–	–	$7.81e{-}1$	–	–	$9.76e{-}1$
c880	–	–	–	–	$9.98e{-}1$	–	–	–
cavlc	–	–	–	–	$7.66e{-}1$	–	–	$8.88e{-}1$
ctrl	0.05	$8.22e{-}1$	–	–	$8.49e{-}1$	–	–	$8.73e{-}1$
dec	184.52	$9.87e{-}1$	–	–	$9.88e{-}1$	–	–	$9.99e{-}1$
i2c	–	–	–	–	$9.98e{-}1$	–	–	–
int2float	4.33	$4.32e{-}1$	–	–	$5.22e{-}1$	–	–	$6.46e{-}1$
max	–	–	–	–	$1.00e{+}0$	–	–	–
priority	–	–	–	–	$1.00e{+}0$	–	–	–
router	–	–	–	–	$2.41e{-}1$	–	–	$3.10e{-}1$

and memory consumption on random and strategic-company formulas.

- The proposed solver `reSSAT` is able to derive non-trivial bounds of satisfying probability, while `DC-SSAT` suffered from timeouts on most of the PEC formulas.

- The minterm-generalization technique is of vital importance to improve the performance of `reSSAT`.

DOI

Chapter 6

Exist-Random Quantified SSAT

In this chapter, we propose a new algorithm motivated by a recent QBF-solving technique for exist-random quantified SSAT formulas, which are also known as *E-MAJSAT* [62]. Most content in this chapter is based on our conference paper [57] published at IJCAI'18.

6.1 Preliminaries

An E-MAJSAT formula Φ has the form $\exists X, \text{ᴙ} Y.\phi(X, Y)$, where X and Y are two disjoint sets of Boolean variables, and $\phi(X, Y)$ is a CNF formula.

DOI

6.1.1 Solving E-MAJSAT with weighted model counting

Given an E-MAJSAT formula $\Phi = \exists X, \mathrm{Ƨ}Y.\phi(X, Y)$ and an assignment τ over X, cofactoring the matrix with τ results in a formula $\phi|_\tau$ referring only to variables in Y. The prefix $\mathrm{Ƨ}Y$ induces a weighting function $\omega : Y \mapsto [0, 1]$ for each variable $y \in Y$, where $\omega(y)$ equals the probability annotated on the randomized quantifier of y. As a result, the conditional satisfying probability $\Pr[\mathrm{Ƨ}Y.\phi|_\tau]$, which equals the weight of the formula $\phi|_\tau$ under the weighting function ω, can be obtained by invoking a weighted model counter. In the following, the invocation to a weighted model counter is expressed by `ComputeWeight`$(\mathrm{Ƨ}Y.\phi|_\tau)$, which returns the conditional satisfying probability $\Pr[\mathrm{Ƨ}Y.\phi|_\tau]$.

6.1.2 Clause selection

Clause selection [43, 90] is a novel technique for QBF solving. Given a CNF formula $\phi(X, Y)$ over are two disjoint variable sets X and Y, we divide each clause $C \in \phi$ into two sub-clauses C^X and C^Y, where C^X (resp. C^Y) consists of the literals whose variables are from X (resp. Y). For example, given a clause $C = (x_1 \vee x_2 \vee y_1 \vee y_2)$, we have $C^X = (x_1 \vee x_2)$ and $C^Y = (y_1 \vee y_2)$. Clearly, $C = C^X \vee C^Y$.

A clause C is said to be *selected* by an assignment τ over X if τ falsifies every literal in C^X; C is said to be *deselected* if τ assigns some literal in C^X to $\top$; C is said to be *undecided* if it is neither selected nor deselected. We also use $\phi|_\tau$ to

DOI

denote the set of clauses selected by an assignment τ over X.

A *selection variable* s_C is introduced for each clause C and defined by $s_C \equiv \neg C^X$. Hence, s_C is an indicator of the selection of clause C. That is, $s_C = \top$ (resp. $s_C = \bot$) indicates C is selected (resp. deselected). Let S be the set of selection variables for clauses in $\phi(X, Y)$. The formula $\psi(X, S) = \bigwedge_{C \in \phi} (s_C \equiv \neg C^X)$ is called a *selection relation* of $\phi(X, Y)$.

Example 6.1. *Consider a formula $\phi(X, Y)$ over two variable sets $X = \{e_1, e_2, e_3\}$ and $Y = \{r_1, r_2, r_3\}$. $\phi(X, Y)$ consists of four clauses:*

$$C_1 : (e_1 \vee r_1 \vee r_2)$$

$$C_2 : (e_1 \vee e_2 \vee r_1 \vee r_2 \vee \neg r_3)$$

$$C_3 : (\neg e_2 \vee \neg e_3 \vee r_2 \vee \neg r_3)$$

$$C_4 : (\neg e_1 \vee e_3 \vee r_3)$$

A set S of selection variables $\{s_1, s_2, s_3, s_4\}$ is introduced for each clause, respectively. The selection relation $\psi(X, S)$ of $\phi(X, Y)$ equals

$$\psi(X, S) = (s_1 \equiv \neg e_1) \wedge (s_2 \equiv \neg e_1 \wedge \neg e_2) \wedge (s_3 \equiv e_2 \wedge e_3) \wedge (s_4 \equiv e_1 \wedge \neg e_3).$$

Consider a complete assignment $\tau_1 = \neg e_1 \neg e_2 \neg e_3$ over X. C_1 and C_2 are selected while C_3 and C_4 are deselected, as can be seen from the selection relation cofactored with τ_1, which results in $\psi(X, S)|_{\tau_1} = s_1 s_2 \neg s_3 \neg s_4$. Consider a partial assignment

DOI

$\tau_2 = \neg e_1 e_3$ over X. It selects C_1, deselects C_4, and leaves C_2 and C_3 undecided. Notice that the two complete assignments $\neg e_1 \neg e_2 e_3$ and $\neg e_1 e_2 e_3$ consistent with τ_2 select $\{C_1, C_2\}$ and $\{C_1, C_3\}$, respectively. The clause C_1 selected by the partial assignment τ_2 lies in the intersection of the sets of clauses selected by the two complete assignments consistent with τ_2.

6.2 Clause-containment learning for E-MAJSAT

Consider an E-MAJSAT formula $\Phi = \exists X, \text{ƺ} Y.\phi$. To obtain the satisfying probability of Φ, it suffices to enumerate every assignment τ over X and calculate the corresponding conditional satisfying probability $\Pr[\Phi|_\tau]$. Clearly, the above brute-force approach is computationally expensive. Motivated by the idea of clause selection discussed above, we propose *clause-containment learning* to prune the search space. The proposed learning technique deduces useful information after each trial of an assignment τ over X. The learnt information is recorded as blocking clauses to avoid wasteful exploration and thus accelerates the search process. The proposed learning technique is based on the following key observation.

Proposition 6.1. *Given an E-MAJSAT formula $\Phi = \exists X, \text{ƺ} Y.\phi(X, Y)$ and two assignments τ_1, τ_2 over X, we have:*

$$(\phi|_{\tau_2} \models \phi|_{\tau_1}) \rightarrow \Pr[\Phi|_{\tau_2}] \leq \Pr[\Phi|_{\tau_1}].$$

Inspired by Proposition 6.1, we propose clause-containment learning based on

DOI

clause selection. After cofactoring ϕ with an arbitrary assignment τ_1 over X, a set of clauses $\phi|_{\tau_1}$ is selected. For any other assignment τ_2 selecting all clauses from $\phi|_{\tau_1}$, i.e., $\phi|_{\tau_1} \subseteq \phi|_{\tau_2}$, we have $\phi|_{\tau_2} \models \phi|_{\tau_1}$. Therefore, $\Pr[\Phi|_{\tau_2}] \leq \Pr[\Phi|_{\tau_1}]$ holds according to Proposition 6.1. Since the satisfying probability $\Pr[\Phi|_{\tau_2}]$ is not greater than $\Pr[\Phi|_{\tau_1}]$, the assignment τ_2 is not worth trying. For all such assignments, they should be blocked after τ_1 has been explored.

The core concept of the clause-containment learning is to avoid every unexplored assignment τ_2 that selects a clause set $\phi|_{\tau_2}$ containing another clause set $\phi|_{\tau_1}$ selected by an explored assignment τ_1. To block the assignment τ_2, we enforce at least one of the clauses in $\phi|_{\tau_1}$ to be deselected. Recall that the selection variable s_C of clause C valuates to $\perp$ if and only if C is deselected. Therefore, the disjunction of the negation of the selection variables for the clauses in $\phi|_{\tau_1}$ is deduced as a learnt clause to record this information. The above idea gives rise to Alg. 5 for E-MAJSAT formulas. (Lines 3, 7, 8, and 11 describe the enhancement techniques of the proposed algorithm, which will be discussed later.)

The algorithm employs two SAT solvers: one works on the matrix $\phi(X, Y)$ of the input formula, and the other works on the selection relation $\psi(X, S)$ for clauses in $\phi(X, Y)$. Using the definition of selection variables, we initialize the selection relation and assert the literals of pure X variables in line 1. If a variable e in X is pure in ϕ, assigning the literals of e to $\top$ deselects the clauses containing e, and does not affect other clauses. Because the conditional satisfying probability is greater if

DOI

Algorithm 5 Solving exist-random quantified SSAT (E-MAJSAT) formulas

Input: $\Phi = \exists X, \text{ƻ} Y.\phi(X, Y)$

Output: $\Pr[\Phi]$

1: $\psi(X, S) := \bigwedge_{C \in \phi} (s_C \equiv \neg C^X) \wedge \bigwedge_{\text{pure } l: \texttt{var}(l) \in X} l$

2: $prob := 0$

3: `s-table` := `BuildSubsumptionTable`(ϕ)

4: **while** (`SAT`(ψ)) **do**

5: $\tau := \psi$.model (discarding the selection variables)

6: **if** (`SAT`$(\phi|_\tau)$) **then**

7: $\tau' :=$ `SelectMinimalClauses`(ϕ, ψ)

8: $\varphi :=$ `RemoveSubsumedClauses`$(\phi|_{\tau'}$, `s-table`$)$

9: $prob := \max\{prob,$ `ComputeWeight`$(\text{ƻ} Y.\varphi)\}$

10: $C_S := \bigvee_{C \in \varphi} \neg s_C$

11: $C_L :=$ `DiscardLiterals`$(\phi, C_S, prob)$

12: **else**

13: $C_L :=$ `MinimalConflicting`(ϕ, τ)

14: **end if**

15: $\psi := \psi \wedge C_L$

16: **end while**

17: **return** $prob$

DOI

less clauses are selected, we can safely assert pure X variables without missing the optimizing solution.

The selection relation is used to select different assignments τ over X. If $\phi|_\tau$ is satisfiable, a weighted model counter is called to compute the conditional satisfying probability $\Pr[\text{⅋}Y.\phi|_\tau]$. The blocking clause C_L derived from the containment-learning technique is then conjoined with ψ to prevent clauses in $\phi|_\tau$ being simultaneously selected again.

On the other hand, suppose $\phi|_\tau$ is unsatisfiable. We can deduce a conjunction of literals from τ responsible for the conflict by using a SAT solver to analyze the conflict [31, 32]. In general, the conjunction of literals may not be minimal, meaning that some literals can be discarded and the conflict remains unaffected. The subroutine `MinimalConflicting` makes the conjunction of literals responsible for the conflict minimal as follows. For each literal l in the conjunction, temporarily drop l and check whether $\phi(X, Y)$ is still unsatisfiable. If it is unsatisfiable, discard l; otherwise, keep l in the conjunction. Repeating the above process for every literal makes the conjunction minimal. Complementing the minimal conjunction of literals yields a learnt clause, which is then conjoined with the selection relation to block assignments that make ϕ unsatisfiable.

When the selection relation becomes unsatisfiable, it indicates that the space spanned by variables X has been completely searched. The algorithm will return the encountered maximum conditional satisfying probability, which equals the satisfying

DOI

probability of the input E-MAJSAT formula.

We illustrate the working of algorithm `SolveEMAJSAT` in the following example.

Example 6.2. *Continuing Example 6.1, we show how Alg. 5 (without the enhancement techniques) solves the E-MAJSAT instance*

$$\Phi = \exists e_1, \exists e_2, \exists e_3, \text{Ꙗ}^{0.5} r_1, \text{Ꙗ}^{0.5} r_2, \text{Ꙗ}^{0.5} r_3 . \phi.$$

Let the first explored assignment τ_1 be $\neg e_1 \neg e_2 \neg e_3$, which selects C_1 and C_2. The algorithm derives $\Pr[\text{Ꙗ}^{0.5} r_1, \text{Ꙗ}^{0.5} r_2, \text{Ꙗ}^{0.5} r_3 . \phi|_{\tau_1}] = 0.75$ by invoking a weighted model counter in line 9. The learnt clause $C_L = (\neg s_1 \vee \neg s_2)$ is conjoined with ψ to prevent C_1 and C_2 being selected simultaneously again.

Suppose the second explored assignment τ_2 is $\neg e_1 e_2 \neg e_3$, which selects C_1. The weighted model counter returns $\Pr[\text{Ꙗ}^{0.5} r_1, \text{Ꙗ}^{0.5} r_2, \text{Ꙗ}^{0.5} r_3 . \phi|_{\tau_2}] = 0.75$, and the learnt clause $C_L = (\neg s_1)$ is conjoined with ψ to prevent C_1 being selected again.

Let the third tried assignment τ_3 be $e_1 e_2 \neg e_3$, which selects C_4. The weighted model counter gives $\Pr[\text{Ꙗ}^{0.5} r_1, \text{Ꙗ}^{0.5} r_2, \text{Ꙗ}^{0.5} r_3 . \phi|_{\tau_3}] = 0.5$, and the learnt clause $C_L = (\neg s_4)$ is conjoined with ψ to prevent C_4 being selected again.

Let the fourth tried assignment τ_4 be $e_1 e_2 e_3$, which selects C_3. The conditional satisfying probability $\Pr[\text{Ꙗ}^{0.5} r_1, \text{Ꙗ}^{0.5} r_2, \text{Ꙗ}^{0.5} r_3 . \phi|_{\tau_4}]$ equals 0.75, and the learnt clause $C_L = (\neg s_3)$ is conjoined with ψ to prevent C_3 being selected again.

Suppose the fifth tried assignment τ_5 is $e_1 \neg e_2 e_3$, which deselects every clause,

DOI

making $\phi|_{\tau_5} = \top$ *and* $\Pr[\text{Ƨ}^{0.5} r_1, \text{Ƨ}^{0.5} r_2, \text{Ƨ}^{0.5} r_3 . \phi|_{\tau_5}] = 1$. *Since there is no selected clause, the learnt clause* C_L *is empty, and the selection relation becomes unsatisfiable after being conjoined with an empty clause. The unsatisfiability of the selection relation reveals that the space spanned by variables* X *has been exhaustively searched, and the algorithm returns the satisfying probability, which equals* 1*, of the E-MAJSAT instance.*

6.2.1 Enhancement techniques

The efficiency of Alg. 5 is greatly affected by the strength of the learnt clauses. We introduce three enhancement methods, *minimal clause selection*, *clause subsumption*, and *partial assignment pruning*, to strengthen the learnt clauses deduced by the proposed learning technique. In Alg. 5, the enhancement techniques are executed by subroutines `SelectMinimalClauses` (in line 7), `RemoveSubsumedClauses` (in line 8), and `DiscardLiterals` (in line 11), to be detailed in the following three parts, respectively.

Minimal clause selection

As discussed before, the selection relation $\psi(X, S)$ is in charge of choosing an assignment τ over variables X and thus selects a set of clauses from the matrix $\phi(X, Y)$. However, the set of selected clauses may not be minimal, meaning that it is possible

DOI

for another assignment τ' to select a set of clauses contained in that selected by τ, i.e., $\phi|_{\tau'} \subset \phi|_{\tau}$. Notice that the length of a learnt clause equals the number of selected clauses. Therefore, selecting fewer clauses gives a stronger learnt clause, as well as a higher conditional satisfying probability.

Starting from a set of initially selected clauses $\phi|_{\tau}$, the first enhancement technique *minimal clause selection* decreases the number of selected clauses by making the set of initially selected clauses minimal as follows. A learnt clause $C_S = \bigvee_{C \in \phi|_{\tau}} \neg s_C$ is conjoined with the selection relation $\psi(X, S)$ to avoid clauses in $\phi|_{\tau}$ being selected simultaneously again. A SAT solver is invoked to solve $\psi \wedge C_S$ under an assumption $\mu = \bigwedge_{C \notin \phi|_{\tau}} \neg s_C$. The assumption μ prevents originally deselected clauses being selected. If $\psi \wedge C_S$ under the assumption μ is satisfied by some assignment τ' over X, the set of clauses selected by τ' must be a proper subset of that selected by τ. On the other hand, if $\psi \wedge C_S$ under the assumption μ is unsatisfiable, then the set of selected clauses is minimal. To make the set of initially selected clauses minimal, the above operation is repeated until the selection relation becomes unsatisfiable.

The subroutine `SelectMinimalClauses` for the technique is described in Alg. 6. The following example illustrates how this technique improves the solving efficiency.

Example 6.3. *Continue Example 6.2. The first tried assignment* $\tau_1 = \neg e_1 \neg e_2 \neg e_3$ *selects* C_1 *and* C_2*. The set of selected clauses is made minimal as follows. The subroutine* `SelectMinimalClauses` *conjoins the selection relation with a learnt clause* $(\neg s_1 \vee \neg s_2)$*, which prevents clauses* C_1 *and* C_2 *being simultaneously selected again.*

DOI

Algorithm 6 Subroutine of Alg. 5: `SelectMinimalClauses`

Input: The matrix $\phi(X, Y)$ and selection relation $\psi(X, S)$

Output: An assignment τ' that selects a minimal set of clauses from ϕ

1: **repeat**

2: $\tau' := \psi$.model (discarding the selection variables)

3: $C_S := \bigvee_{C \in \phi|_{\tau'}} \neg s_C$

4: $\psi := \psi \wedge C_S$

5: $\mu := \bigwedge_{C \notin \phi|_{\tau'}} \neg s_C$

6: **until** (`UNSAT`$(\psi|_\mu)$)

7: **return** τ'

The satisfiability of the selection relation $\psi \wedge (\neg s_1 \vee \neg s_2)$ is tested under an assumption $\neg s_3 \neg s_4$, which avoids the originally deselected clause C_3 and C_4 being selected. The formula is satisfied by the assignment $\tau_2 = \neg e_1 e_2 \neg e_3$, which only selects C_1. By repeating the above operation again, the assignment $\tau_5 = e_1 \neg e_2 e_3$ is found, since it selects no clause. The conditional satisfying probability under τ_5, which equals 1, is derived without invoking a weighted model counter. Compared to 6.2 where five assignments over X were explored, the algorithm with the minimal clause selection technique finds the optimal assignment τ_5 with at most two SAT calls, thus greatly improves the computation efficiency.

DOI

Clause subsumption

The second enhancement technique, named *clause subsumption*, decreases the length of a learnt clause via examining the subsumption relation among the selected clauses. Recall that clause C_1 *subsumes* clause C_2 if every literal appears in C_1 also appears in C_2. Consider a CNF formula $C_1 \wedge C_2$ with C_2 subsumed by C_1. It can be simplified to C_1 because C_2 is implied by C_1 due to the subsumption relation.

The subsumption relation among sub-clauses consisting of variables in Y is constructed as a lookup table by the subroutine `BuildSubsumeTable`. The subroutine `RemoveSubsumedClauses` simplifies the set of selected clauses $\phi|_\tau$ by removing subsumed clauses. A clause C is removed from the set of selected clauses $\phi|_\tau$ if C^Y is subsumed by other selected sub-clauses. We emphasize that, without cofactoring ϕ with the assignment τ, the original clauses may not have the subsumption relation. Cofactoring ϕ with the assignment τ over variables in X induces the subsumption relation between sub-clauses consisting of variables in Y.

The procedure `RemoveSubsumedClauses` for the technique is outlined in Alg. 7. It takes a set of selected clauses and a lookup table of subsumption relation as input, and removes every subsumed clause via subroutine `CheckNotSubsumed`. The following example explains how the subsumption relation shortens a learnt clause.

Example 6.4. *Continue Example 6.2. Recall that the first tried assignment is* $\tau_1 = \neg e_1 \neg e_2 \neg e_3$. *It selects* C_1 *and* C_2. *Because* C_1^Y *subsumes* C_2^Y, C_2 *is removed from*

DOI

Algorithm 7 Subroutine of Alg. 5: `RemoveSubsumedClauses`

Input: The selected clauses $\phi|_\tau$ and a subsumption table `s-table`

Output: A simplified clause set φ without subsumed clauses

1: $\varphi := \top$
2: **for** $(C \in \phi|_\tau)$ **do**
3: **if** (`CheckNotSubsumed`$(C,$ `s-table`$)$) **then**
4: $\varphi := \varphi \wedge C$
5: **end if**
6: **end for**
7: **return** φ

$\phi|_{\tau_1}$, yielding $\varphi = C_1$. A weighted model counter computes $\Pr[\text{Ꙗ}^{0.5}Y.\varphi] = 0.75$, and the learnt clause $C_L = (\neg s_1)$ is conjoined with ψ to prevent C_1 being selected again. Compared to Example 6.2, with the help of clause subsumption, the learnt clause $(\neg s_1)$ deduced from τ_1 is stronger than its counterpart $(\neg s_1 \vee \neg s_2)$ in Example 6.2, and therefore avoids a fruitless trail of the assignment $\tau_2 = \neg e_1 e_2 \neg e_3$.

Partial assignment pruning

To illustrate the third enhancement technique *partial assignment pruning*, we first take a closer look at a learnt clause deduced by the proposed clause-containment learning. Given a matrix $\phi(X, Y)$ and an assignment τ over X, a learnt clause is a disjunction of the negated selection variables of the selected clauses. For each

DOI

selected clause C, if the selection variable s_C is substituted by its definition $s_C \equiv \neg C^X$, the learnt clause C_L becomes a disjunction of the sub-clauses C^X, i.e., $C_L = \bigvee_{C \in \phi|_\tau} C^X$.

For instance, in Example 6.2, the learnt clause deduced from the assignment $\tau_1 = \neg e_1 \neg e_2 \neg e_3$, which selects clauses C_1 and C_2, is $(\neg s_1 \vee \neg s_2) = (e_1 \vee e_2)$, and the current maximum satisfying probability equals 0.75. This learnt clause blocks two assignments, τ_1 and $\neg e_1 \neg e_2 e_3$. The assignment $\neg e_1 \neg e_2 e_3$ is blocked because it selects clauses C_1, C_2, and C_4, and clauses C_1 and C_2 have been selected previously.

A learnt clause can be strengthened if some literal in the clause is discarded. In the above example, the learnt clause $(e_1 \vee e_2)$ can be strengthened by discarding literal e_2. The resulted learnt clause (e_1) blocks any assignment assigning e_1 to $\perp$. To see why these assignments are blocked, observe that assigning e_1 to $\perp$ selects clause C_1. Although the set $\{C_1\}$ does not contain the set $\{C_1, C_2\}$, the conditional satisfying probability is bounded from above by the probability of the sub-clause $(r_1 \vee r_2)$, which equals 0.75. Since the current maximum satisfying probability equals 0.75 already, it is fruitless to explore assignments whose conditional satisfying probabilities are no greater than 0.75. Therefore, e_1 is forced to be $\top$.

On the other hand, literal e_1 cannot be discarded from the learnt clause $(e_1 \vee e_2)$. If e_1 is discarded, the resulted learnt clause (e_2) blocks any assignment assigning e_2 to $\perp$. However, assigning e_2 to $\perp$ selects no clause, and hence the upper bound of the conditional satisfying probability equals 1. Since 1 is greater than the current

DOI

maximum satisfying probability 0.75, we have to explore assignments that map e_2 to $\perp$. In fact, there exists an assignment $e_1 \neg e_2 \neg e_3$ whose conditional satisfying probability equals 1, greater than the current maximum satisfying probability. As a result, e_1 cannot be discarded.

From the above illustration, we observe that a learnt clause can be strengthened as follows. First, temporarily discard some literal l from a learnt clause. Second, invoke a weighted model counter to compute the conditional satisfying probability contributed by the selected clauses. Third, compare the probability to the current maximum satisfying probability. If the probability is no greater, discard literal l; otherwise, keep l in the learnt clause. Fourth, repeat the above steps for other literals.

The subroutine `DiscardLiterals` of the technique to discard literals from a learnt clause is outlined in Alg. 8. It substitutes selection variables in a learnt clause C_S by their definitions in line 1. Then it iterates through each literal $l \in C_L$ and invokes a weighted model counter to check whether literal l can be discarded in lines 2 to 5. Notice that the current maximum satisfying probability is *not* updated by the value obtained by weighted model counting, because the value only reflects an upper bound for satisfying probabilities under the partial assignment ν.

Example 6.5. *Continue Example 6.2. As discussed above, the learnt clause* $(e_1 \vee e_2)$ *deduced from the assignment* $\tau_1 = \neg e_1 \neg e_2 \neg e_3$ *is strengthen to* (e_1) *by the partial assignment pruning technique, and thus prevents a fruitless trail of the assignment*

DOI

Algorithm 8 Subroutine of Alg. 5: `DiscardLiterals`

Input: The matrix ϕ, a learnt clause C_S, and the current maximum *prob*

Output: A strengthened learnt clause C_L

1: $C_L := \bigvee_{s_C \in C_S} C^X$
2: **for** $(l \in C_L)$ **do**
3: $\quad \nu := \bigwedge_{k \in C_L \setminus \{l\}} \neg k$
4: $\quad$ **if** $(\texttt{ComputeWeight}(\text{Ꙗ}Y.\phi|_\nu) \leq prob)$ **then**
5: $\quad\quad C_L := C_L \setminus \{l\}$
6: $\quad$ **end if**
7: **end for**
8: **return** C_L

$\tau_2 = \neg e_1 e_2 \neg e_3$.

To summarize, the above three enhancement techniques are designed based on different reasoning strategies: the minimal clause selection technique strengthens the learnt clause by invoking additional SAT calls to solve the selection relation to select a minimal set of clauses; the clause subsumption technique builds a lookup table to record the subsumption relation and removes subsumed clauses; the partial assignment pruning technique utilizes the current maximum conditional satisfying probability, and invokes extra calls to a weighted model counter to test whether it is feasible to discard some literal in a learnt clause. Benefiting from the three enhancement techniques, the efficiency of the proposed algorithm is greatly improved, as will be shown in our experiments.

DOI

We emphasize two more strengths of the proposed algorithm. First, during the computation, the proposed algorithm keeps deriving lower bounds for the satisfying probability, and the bounds gradually converge to the final answer. Therefore, in contrast to previous DPLL-based methods [67–69], the proposed algorithm can be easily modified to solve *approximate SSAT* by returning the greatest encountered lower bound upon timeout. Second, the proposed algorithm is efficient in memory usage, since it stores the learnt information compactly via selection variables, and the weighted model counting is invoked on selected clauses, whose sizes are typically much smaller than that of original matrix.

6.2.2 Implementation details

We discuss some details about our implementation of the proposed algorithm. Alg. 5 involves satisfiability testing and weighted model counting. In principle, any SAT solver and exact weighted model counter can be plugged into the procedure. Therefore, Alg. 5 may benefit directly from the advancement of SAT solving and model counting.

Specifically in our implementation, we adopt `Minisat-2.2` [31] for SAT solving. We experimented with `Cachet` [95, 96] for weighted model counting, but the overall performance was not satisfactory. Because the clauses selected by different assignments typically share a portion of identical sub-formulas, the results of these sub-formulas should be cached to avoid repeated computations. Ideally, this im-

DOI

provement can be achieved by tightly integrating `Cachet` into our implementation, instead of treating it as a black box. However, we resort to *binary decision diagram* (BDD) for our weighted model counting to achieve the *formula caching* effect.

The selected clauses in a formula are first transformed into a two-level circuit by OR-ing literals in each clause and AND-ing the outputs of each OR gate. A BDD is then constructed based on the resulted circuit. Weighted model counting of the original formula is done by traversing the BDD [54]. We use the well-developed BDD package `CUDD` [101], which caches BDD nodes during building BDDs for different formulas. Thus the formula caching effect, as well as garbage collection which removes unreferenced nodes, is automatically achieved by `CUDD`.

Our implementation was integrated in the `ABC` [11]environment, which provides all the facilities including SAT solving, circuit construction, and BDD computation mentioned above.

6.3 Evaluation

We evaluated the proposed Alg. 5 against the state-of-the-art DPLL-based SSAT solver `DC-SSAT` [67] over both random k-CNF and application formulas. The proposed algorithm is implemented in the `C++` language inside the `ABC` [11] environment. The SAT solver `MiniSat`-2.2 [31] is used to answer satisfiability queries. For weighted model counting, we tried `Cachet` [95, 96], but the overall performance was not satis-

DOI

factory. Instead, we resorted to a well-developed BDD package `CUDD` [101]. Weight computation of a formula is fulfilled via a classic approach [27] that traverses the BDD of a formula and computes the satisfying probabilities of the BDD nodes. Our prototyping implementation[1] is named `erSSAT`. A bare version of `erSSAT` without the enhancement techniques is called `erSSAT-b` in the experiments. We used commit `2ff8e74` of branch `master` in the experiments.

6.3.1 Benchmark set

The SSAT instances in the evaluation are hosted in a publicly available database[2]. We used commit `ea9fbae` of branch `master` in the experiments.

Random k-CNF formulas

We generated random k-CNF formulas by `CNFgen` [53]. A collection of 700 formulas were generated with k, i.e., the number of literals in a clause, taking values from $\{3, 4, 5, 6, 7, 8, 9\}$, the number of variables taking values from $\{10, 20, 30, 40, 50\}$, and clause-to-variable ratio taking values from $\{k-1, k, k+1, k+2\}$. Five formulas were sampled for each parameter combination. To convert the propositional formulas into E-MAJSAT formulas, the first half of the variables are existentially quantified, and the rest are randomly quantified with probability 0.37.

[1] Available at: https://github.com/NTU-ALComLab/ssatABC

[2] Available at: https://github.com/NTU-ALComLab/ssat-benchmarks

DOI

Application formulas

Table 6.1: The families of the application formulas

Family	Description	Number
Toilet-A	Adapted from exist-forall QBFs [82]	77
Conformant	Adapted from exist-forall QBFs [82]	24
Sand-Castle	A probabilistic planning problem [68]	25
Max-Count	Adapted from maximum model counting [35]	26
MPEC	Maximum probabilistic equivalence checking	60

We collected five families of application formulas for evaluation. Their descriptions and the numbers of the instances in each family are summarized in Table 6.1. The first two families, *Toilet-A* and *Conformant*, were adapted from exist-forall-exist QBFs [82]. We converted the QBFs into exist-random-exist quantified SSAT formulas by replacing their universal quantifiers with randomized ones with probabilities 0.5. The third family *Sand-Castle* is a probabilistic conformant planning domain. The problem can be encoded as E-MAJSAT formulas [68]. The family *Max-Count* models the problems of maximum satisfiability, quantitative information flow, and program synthesis with maximum model counting [35]. We represented the maximum model counting instances as E-MAJSAT formulas. The last family *MPEC* consists of formulas that analyze the maximum probability of a probabilistic circuit to produce erroneous outputs, as discussed in Chapter 4.

DOI

6.3.2 Experimental setup

Our experiments were performed on a machine with one 2.2 GHz CPU (Intel Xeon Silver 4210) with 40 processing units and 134 616 MB of RAM. The operating system was Ubuntu 20.04 (64 bit), running Linux 5.4. The programs were compiled with `g++ 9.3.0`. Each SSAT-solving task was limited to a CPU core, a CPU time of 15 min, and a memory usage of 15 GB. To achieve reliable benchmarking, we used a benchmarking framework `BenchExec`[3] [9].

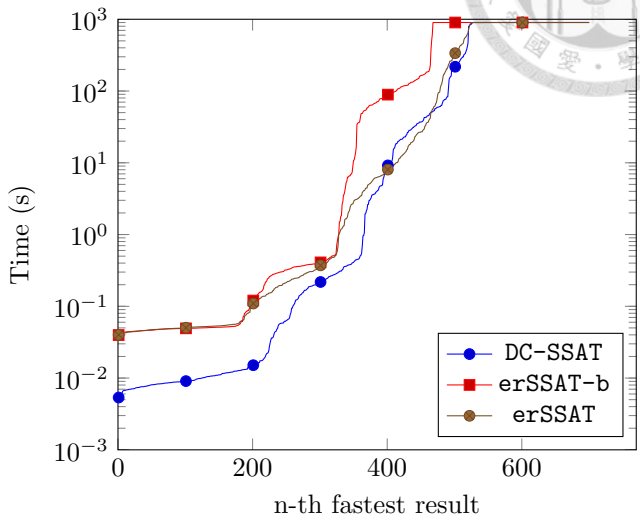
6.3.3 Results

Random k-CNF formulas

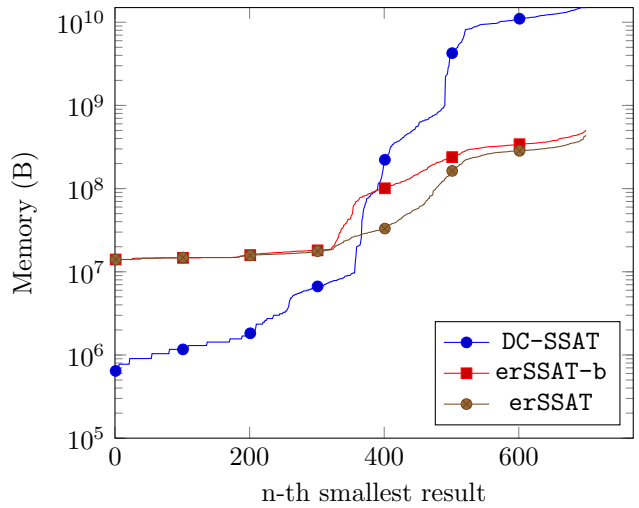
Fig. 6.1 shows the quantile plots regarding CPU time and memory usage of the SSAT instances derived from the random k-CNF formulas. A data point (x, y) in a quantile plot indicates that there are x formulas processed by the respective algorithm within a resource constraint of y. In Fig. 6.1a, we observe that `erSSAT` solved a similar amount of formulas as `DC-SSAT` did. Moreover, the enhancement techniques improve the performance of `erSSAT` a lot, as can be seen from the huge difference between `erSSAT` and `erSSAT-b`. On the other hand, Fig. 6.1b shows that `DC-SSAT` used much more memory than `erSSAT`. This can be attributed to the

[3] Available at: https://github.com/sosy-lab/benchexec

DOI



(a) CPU time



(b) Memory usage

Figure 6.1: Quantile plots of random k-CNF formulas

DOI

Table 6.2: Summary of the results for 212 application formulas

Algorithm	DC-SSAT	erSSAT	erSSAT-b
Solved formulas	78	59	65
Toilet-A	44	38	46
Conformant	1	2	1
Sand-Castle	22	13	14
Max-Count	3	3	1
MPEC	8	3	3
Timeouts	85	141	129
Out of memory	38	0	0
Other inconclusive	11	12	18

subformula caching of DC-SSAT. Instead, erSSAT only builds BDDs for cofactored formulas, which confined its memory footprint.

Application formulas

The solving results of the application formulas are summarized in Table 6.2. For each compared approach, the numbers of its exactly solved formulas, timeouts, out of memory, and other inconclusive situations are reported. To study the solving performance regarding different kinds of formulas, we further report the numbers of exactly solved formulas per family. Observe that DC-SSAT exactly solved the most

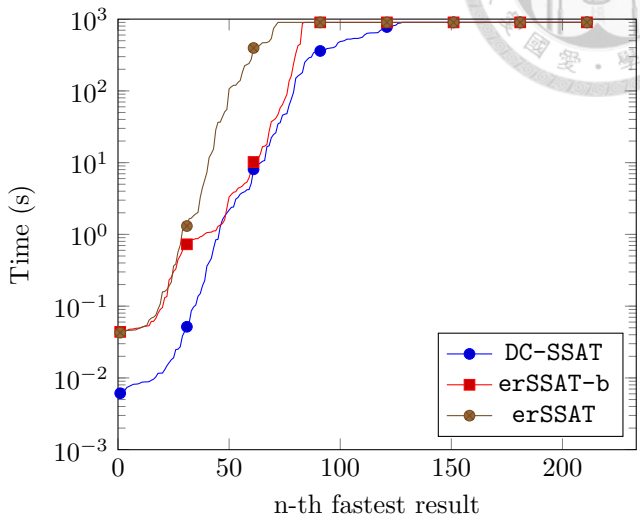
DOI

formulas. Its advantage mainly comes from family *Sand-Castle*, where it solved 22 formulas, but `erSSAT` and `erSSAT-b` only solved 13 and 14 formulas, respectively. We will analyze why the proposed clause-containment learning is not suitable for this family later. To our surprise, the proposed enhancement techniques seem not very useful on the evaluated application formulas. They even worsened the performance over formulas from the family *Toilet-A*. While `erSSAT` and `erSSAT-b` suffered from more timeouts than `DC-SSAT`, they did not run out of memory for any formula. Instead, `DC-SSAT` tends to consume a lot of memory, because it memorizes many subformulas.

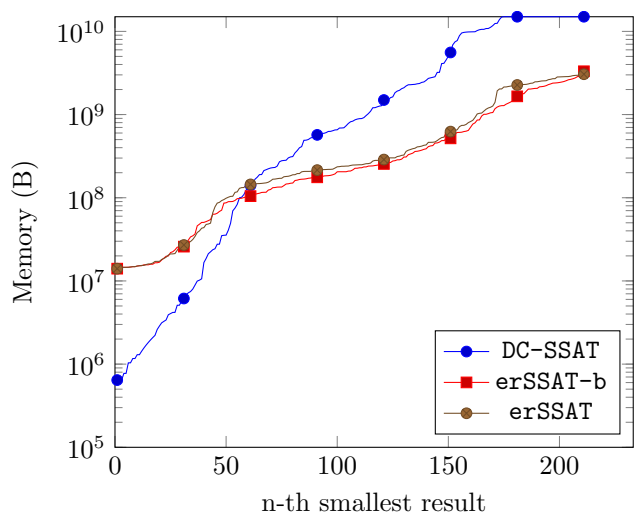
Fig. 6.2 shows the quantile plots of the application SSAT formulas. We can see that the enhancement techniques affected not only the effectiveness of `erSSAT` but also its efficiency from Fig. 6.2a. This phenomenon indicates that the additional effort spent to strengthen a learnt clause is not worthy. The reason behind this phenomenon will be inspected in the following.

To further examine the suitability of the enhancement techniques, we demonstrate the scatter plots with `erSSAT` in y-axis and compared approaches in x-axis in Fig. 6.3. A data point (x, y) in the plots indicates that there is a formula processed by both `erSSAT` and a compared approach, while `erSSAT` took a CPU time of y seconds and the other approach took a CPU time of x seconds. From Fig. 6.3a, we find that the enhancement techniques did improve the solving of some formulas, but more often they were an overhead to `erSSAT-b`. Fig. 6.3b also shows that

DOI



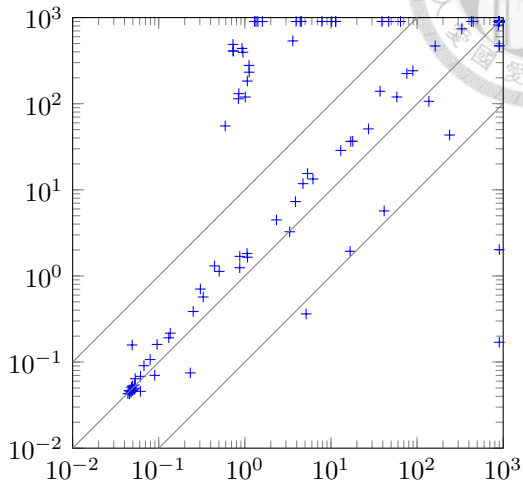
(a) CPU time



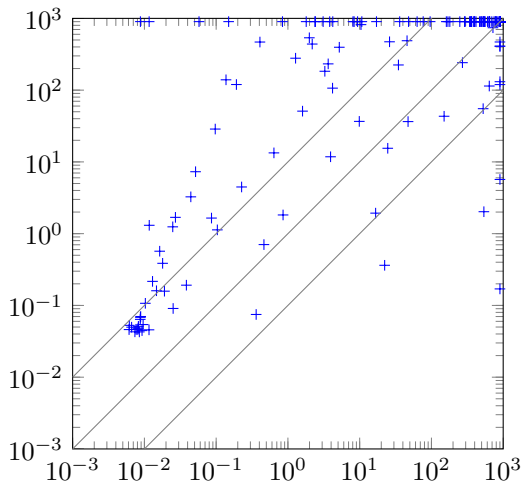
(b) Memory usage

Figure 6.2: Quantile plots of application formulas

DOI



(a) `erSSAT-b`



(b) `DC-SSAT`

Figure 6.3: Run-time scatter plots of application formulas with `erSSAT` in y-axis and compared approaches in x-axis

DOI

DC-SSAT was more efficient to exactly solve formulas than erSSAT over the evaluated application formulas.

Clause-containment learning over the *Sand-Castle* problem

As erSSAT and erSSAT-b did not solve formulas from the probabilistic planning domain *Sand-Castle* quite well, we look into the problem and discuss our findings here. The *Sand-Castle* problem [68] describes an agent who wants to build a sand castle on a beach. The agent has two actions to choose from: digging a moat or erecting a castle. A moat protects a castle from the water and increases the probability to successfully build a castle. The agent must take a unique action at every stage. Under the settings of conformant planning, the agent must decide its strategy beforehand and does not have access to internal states (whether a moat has been digged or a castle has been erected) during the execution. Given a finite number of stages, the problem asks to compute a strategy to maximize the chance of successfully building a castle at the last stage. The agent's actions are encoded with existentially quantified variables, and the nondeterminism in the state-transition mechanism is encoded with randomly quantified variables.

A formula that encodes the *Sand-Castle* problem with n stages has the form:

$$\phi = \bigwedge_{i=1}^{n} \phi_d^{(i)} \wedge \phi_e^{(i)}, \tag{6.1}$$

where ϕ_d and ϕ_e are sets of clauses used to represent the state-transition mechanism when the agent chooses to *dig* a moat or *erect* a castle, respectively. As the state-

DOI

transition mechanism is the same for every stage except for the variables recording the internal state, we use the superscripts to indicate the stage indices. We found that the *Sand-Castle* problem has the following property: the set of clauses $\phi_d^{(i)}$ (resp. $\phi_e^{(i)}$) will be selected if and only if the agent chooses to dig a moat (resp. erect a castle) at stage i. In other words, each strategy of the agent (i.e., an assignment to the existentially quantified variables) will select a distinct set of clauses. Recall that the proposed clause-containment principle aims at blocking assignments selecting a superset of clauses that has been selected by a previously explored assignment. As a result, a learnt clause constructed based on this principle can only block the current assignment itself, which means that `erSSAT-b` degenerates to merely brute-force search. This theoretic reasoning is confirmed by the solving statistics (visible from the log files), which show that `erSSAT-b` invoked $2^n - 1$ model-counting queries. For `erSSAT`, the situation is worse due to partial assignment pruning, which invokes additional model-counting queries to strengthen a learnt clause. From the log files, we found that `erSSAT` invoked twice numbers of model-counting queries than `erSSAT-b`, because it had an additional trial but always ended in vain.

On the other hand, the structure of the *Sand-Castle* formulas favors `DC-SSAT`, as the subformulas are essentially the same across the stages. It is not surprising the formula caching and divide-and-conquer method works well with these formulas.

DOI

Approximate solving

Recall that the proposed Alg. 5 solves an SSAT formula in a converging manner. Instead of computing the exact satisfying probability at once, it keeps deriving lower bounds of the satisfying probability of a formula. This characteristic integrates exact and approximate solving into one approach. In the following, we study the approximation ability of `erSSAT`. We choose families *Conformant*, *Max-Count*, and *MPEC* for detailed investigation, because all of the compared approaches ran out of CPU time or memory over most of their formulas.

Tables 6.3, 6.4, and 6.5 show the approximation results over the above three families, respectively. For `DC-SSAT`, the CPU time and exact satisfying probability are reported. For `erSSAT` and `erSSAT-b`, in addition to the CPU and exact satisfying probability, the tightest lower bound and the time elapsed to derive the lower bound are also shown in the tables. A formula is not shown in the tables if none of the approaches can solve it or derive a non-trivial lower bound for it.

As can be observed from the tables, `erSSAT` was able to derive tight lower bounds for formulas from these families, while `DC-SSAT` suffered from timeouts over most of them. The approximation ability of `erSSAT` makes it useful for large formulas, which cannot be exactly solved by the state-of-the-art approaches.

The above results on the random and application formulas suggest that:

DOI

- The proposed solver `erSSAT` achieves a similar performance as `DC-SSAT` in terms of CPU time and outperforms `DC-SSAT` in terms of memory consumption on random formulas.

- The proposed solver `erSSAT` is not as good as `DC-SSAT` at exactly solving the application formulas, which can be attributed to the overhead caused by the enhancement techniques.

- The proposed solver `erSSAT` is good at deriving tight lower bounds for large formulas. This approximation ability is especially valuable when the size of a formula is beyond the capability of the state-of-the-art exact solver.

To sum up, our evaluation results demonstrate the unique value of the proposed clause-containment learning.

DOI

Table 6.3: Results of solving the *Conformant* family

	DC-SSAT		erSSAT				erSSAT-b			
FORMULA	T (s)	Pr	T (s)	Pr	LB	T-LB (s)	T (s)	Pr	LB	T-LB (s)
blocks_enc_2_b3_ser–opt-9_	–	–	–	–	$4.38e{-1}$	201.11	–	–	$3.13e{-1}$	76.80
blocks_enc_2_b4_ser–opt-26_	–	–	–	–	$4.38e{-1}$	290.22	–	–	$4.38e{-1}$	252.27
cube_c3_ser–opt-6_	16.68	$1.00e{+0}$	1.93	$1.00e{+0}$	–	–	16.70	$1.00e{+0}$	–	–
cube_c5_ser—14_	–	–	–	–	$1.80e{-1}$	281.72	–	–	$1.64e{-1}$	431.66
cube_c5_ser–opt-15_	–	–	–	–	$1.80e{-1}$	215.60	–	–	$4.14e{-1}$	715.65
cube_c7_ser—23_	–	–	–	–	$3.38e{-1}$	516.06	–	–	$3.38e{-1}$	185.91
cube_c7_ser–opt-24_	–	–	–	–	$3.44e{-1}$	680.18	–	–	$3.38e{-1}$	388.99
cube_c9_par—10_	–	–	–	–	$2.90e{-1}$	213.93	–	–	$2.94e{-1}$	416.43
cube_c9_par–opt-11_	–	–	–	–	$2.89e{-1}$	220.44	–	–	$2.93e{-1}$	868.66
emptyroom_e3_par–opt-10_	–	–	–	–	$1.25e{-1}$	512.56	–	–	$1.56e{-1}$	864.63
emptyroom_e3_ser—19_	–	–	–	–	$1.88e{-1}$	611.64	–	–	$1.25e{-1}$	215.96
emptyroom_e3_ser–opt-20_	–	–	–	–	$9.38e{-2}$	218.59	–	–	$1.41e{-1}$	700.72
emptyroom_e4_par—21_	–	–	–	–	$3.91e{-3}$	157.53	–	–	$1.95e{-2}$	358.14
emptyroom_e4_par–opt-22_	–	–	–	–	$3.91e{-3}$	264.68	–	–	$1.56e{-2}$	213.01
emptyroom_e4_ser–opt-44_	–	–	–	–	$3.91e{-3}$	456.61	–	–	$1.56e{-2}$	705.08
ring_r3_ser–opt-8_	–	–	468.99	$1.00e{+0}$	–	–	–	–	$7.89e{-1}$	119.88
ring_r4_ser–opt-11_	–	–	–	–	$4.96e{-1}$	523.06	–	–	$7.19e{-1}$	491.09

DOI

Table 6.4: Results of solving the *Max-Count* family

Formula	DC-SSAT			erSSAT				erSSAT-b		
	T (s)	Pr	T (s)	Pr	LB	T-LB (s)	T (s)	Pr	LB	T-LB (s)
MaxSAT-keller4-1212.clq.wcnf	–	–	0.17	9.76e−1	–	–	–	–	9.65e−1	0.12
QIF-CVE-2007-2875	–	–	2.02	1.00e+0	–	–	–	–	9.54e−7	5.17
QIF-backdoor-2x16-8	0.01	1.53e−5	–	–	5.96e−8	0.16	–	–	5.96e−8	0.15
QIF-backdoor-32-24	0.01	1.00e+0	0.05	1.00e+0	–	–	0.05	1.00e+0	–	–
QIF-bin-search-16	–	–	–	–	1.95e−3	113.67	–	–	1.22e−4	141.96
QIF-reverse	–	–	–	–	5.96e−7	546.68	–	–	9.54e−7	432.36
QIF-reverse2	–	–	–	–	2.98e−7	200.88	–	–	9.54e−7	542.40
SyGuS-sign	528.35	1.00e+0	–	–	–	–	–	–	–	–

DOI

Table 6.5: Results of solving the *MPEC* family

	DC-SSAT			erSSAT				erSSAT-b		
FORMULA	T (s)	Pr	T (s)	Pr	LB	T-LB (s)	T (s)	Pr	LB	T-LB (s)
c1355-0.01	–	–	–	–	4.14e−1	675.81	–	–	6.56e−1	41.01
c1908-0.01	48.04	4.14e−1	–	–	3.18e−1	587.05	–	–	4.14e−1	22.27
c2670-0.01	–	–	–	–	4.87e−1	146.46	–	–	5.51e−1	5.74
c3540-0.01	–	–	–	–	–	–	–	–	5.51e−1	72.00
c432-0.01	–	–	–	–	2.34e−1	141.33	–	–	2.34e−1	2.91
c499-0.01	–	–	–	–	4.14e−1	33.32	–	–	4.14e−1	0.38
c880-0.01	–	–	–	–	3.30e−1	4.14	–	–	3.30e−1	6.17
cavlc-0.01	0.15	5.42e−1	–	–	–	–	–	–	–	–
ctrl-0.01	0.01	2.34e−1	0.07	2.34e−1	–	–	0.09	2.34e−1	–	–
ctrl-0.10	0.06	8.65e−1	–	–	–	–	–	–	–	–
dec-0.01	0.01	6.56e−1	0.04	6.56e−1	–	–	0.05	6.56e−1	–	–
dec-0.10	180.51	9.88e−1	–	–	–	–	–	–	–	–
i2c-0.01	–	–	–	–	–	–	–	–	7.21e−1	175.38
int2float-0.01	0.01	2.34e−1	1.31	2.34e−1	–	–	0.44	2.34e−1	–	–
int2float-0.10	4.22	9.01e−1	–	–	–	–	–	–	–	–
priority-0.01	–	–	–	–	4.45e−1	139.73	–	–	5.89e−1	556.81
router-0.01	–	–	–	–	1.25e−1	0.36	–	–	1.25e−1	0.38

DOI

Chapter 7

Dependency SSAT

In this chapter, we lift SSAT to the NEXPTIME-complete complexity class by formulating *dependency SSAT* (DSSAT). Most content in this chapter is based on our conference paper [55] published at AAAI'21.

7.1 Preliminaries

7.1.1 Dependency quantified Boolean formula

DQBF is formulated as *multiple-person alternation* by Peterson and Reif [87]. In contrast to the *linearly ordered* quantification used in QBF, i.e., an existentially quantified variable depends on all of its preceding universally quantified variables, the quantification structure in DQBF is extended with *Henkin quantifiers*, where

DOI

the dependency of an existentially quantified variable is explicitly specified.

A DQBF Φ over a set $V = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ of variables is of the form:

$$\Phi = \forall x_1, \ldots, \forall x_n, \exists y_1(D_{y_1}), \ldots, \exists y_m(D_{y_m}).\phi(x_1, \ldots, x_n, y_1, \ldots, y_m),$$

where each $D_{y_j} \subseteq \{x_1, \ldots, x_n\}$ denotes the set of universally quantified variables that y_j depends on, and Boolean formula ϕ over V is quantifier-free. We denote the set $\{x_1, \ldots, x_n\}$ (resp. $\{y_1, \ldots, y_m\}$) of universally (resp. existentially) quantified variables of Φ by $V_\Phi^\forall$ (resp. $V_\Phi^\exists$).

A DQBF Φ is satisfiable if for each variable y_j, there exists a Boolean function $f_j : \mathcal{A}(D_{y_j}) \mapsto \mathbb{B}$, such that matrix ϕ becomes a tautology over $V_\Phi^\forall$ after substituting variables in $V_\Phi^\exists$ with their respective Boolean functions. The set of Boolean functions $\mathcal{F} = \{f_1, \ldots, f_m\}$ is called a set of *Skolem* functions for Φ. In other words, Φ is satisfied by $\mathcal{F}$ if the following QBF valuates to $\top$:

$$\forall x_1, \ldots, \forall x_n.\phi(x_1, \ldots, x_n, f_1, \ldots, f_m), \tag{7.1}$$

where $\phi(x_1, \ldots, x_n, f_1, \ldots, f_m)$ represents the formula obtained by substituting existentially quantified variables in ϕ with their respective Skolem functions. We extend the notation for cofactors and denote $\phi(x_1, \ldots, x_n, f_1, \ldots, f_m)$ by $\phi|_\mathcal{F}$. The satisfiability problem of DQBF is NEXPTIME-complete [88].

DOI

7.1.2 Decentralized POMDP

Dec-POMDP is a formalism for multi-agent systems under uncertainty and with partial information. Its computational complexity is NEXPTIME-complete [8]. In the following, we briefly review the definition, optimality criteria, and value function of Dec-POMDP from the literature [84].

A Dec-POMDP is specified by a tuple $\mathcal{M} = (I, S, \{A_i\}, T, \rho, \{O_i\}, \Omega, \Delta_0, h)$, where $I = \{1, \ldots, n\}$ is a finite set of n agents, S is a finite set of states, A_i is a finite set of actions of Agent i, $T : S \times (A_1 \times \cdots \times A_n) \times S \mapsto [0, 1]$ is a transition distribution function with $T(s, \vec{a}, s') = \Pr[s'|s, \vec{a}]$, the probability to transit to state s' from state s after taking actions $\vec{a}$, $\rho : S \times (A_1 \times \cdots \times A_n) \mapsto \mathbb{R}$ is a reward function with $\rho(s, \vec{a})$ giving the reward for being in state s and taking actions $\vec{a}$, O_i is a finite set of observations for Agent i, $\Omega : S \times (A_1 \times \cdots \times A_n) \times (O_1 \times \cdots \times O_n) \mapsto [0, 1]$ is an observation distribution function with $\Omega(s', \vec{a}, \vec{o}) = \Pr[\vec{o}|s', \vec{a}]$, the probability to receive observation $\vec{o}$ after taking actions $\vec{a}$ and transiting to state s', $\Delta_0 : S \mapsto [0, 1]$ is an initial state distribution function with $\Delta_0(s) = \Pr[s^0 \equiv s]$, the probability for the initial state s^0 being state s, and h is a planning horizon, which we assume finite in this work.

Given a Dec-POMDP $\mathcal{M}$, we aim at maximizing the expected cumulative reward $\mathbb{E}[\sum_{t=0}^{h-1} \rho(s^t, \vec{a}^t)]$ through searching an optimal *joint policy* for a team of agents. Specifically, a *policy* π_i of Agent i is a mapping from the agent's *observation history*,

DOI

i.e., a sequence of observations $\underline{o_i^t} = o_i^0, \ldots, o_i^t$ received by Agent i, to an action $a_i^{t+1} \in A_i$. A joint policy for the team of agents $\vec{\pi} = (\pi_1, \ldots, \pi_n)$ maps the agents' joint observation history $\vec{\underline{o}}^t = (\underline{o_1^t}, \ldots, \underline{o_n^t})$ to actions $\vec{a}^{t+1} = (\pi_1(\underline{o_1^t}), \ldots, \pi_n(\underline{o_n^t}))$. We shall focus on deterministic policies only, as it is shown that every Dec-POMDP with a finite planning horizon has a deterministic optimal joint policy [85].

The quality of a joint policy $\vec{\pi}$ is measured by its expected cumulative reward. The *value* of a joint policy is hence defined to be $\mathbb{E}[\sum_{t=0}^{h-1} \rho(s^t, \vec{a}^t) | \Delta_0, \vec{\pi}]$. The *value function* V^π can be computed in a recursive manner, where for $t = h - 1$,

$$V^\pi(s^{h-1}, \vec{\underline{o}}^{h-2}) = \rho(s^{h-1}, \vec{\pi}(\vec{\underline{o}}^{h-2})),$$

and for $t < h - 1$,

$$V^\pi(s^t, \vec{\underline{o}}^{t-1}) = \rho(s^t, \vec{\pi}(\vec{\underline{o}}^{t-1})) + \sum_{s^{t+1} \in S} \sum_{\vec{o}^t \in \vec{O}} \Pr[s^{t+1}, \vec{o}^t | s^t, \vec{\pi}(\vec{\underline{o}}^{t-1})] V^\pi(s^{t+1}, \vec{\underline{o}}^t). \quad (7.2)$$

The probability $\Pr[s^{t+1}, \vec{o}^t | s^t, \vec{\pi}(\vec{\underline{o}}^{t-1})]$ is the product of the transition probability $T(s^t, \vec{\pi}(\vec{\underline{o}}^{t-1}), s^{t+1})$ and the observation probability $\Omega(s^{t+1}, \vec{\pi}(\vec{\underline{o}}^{t-1}), \vec{o}^t)$. Eq. (7.2) is also called the *Bellman equation* of Dec-POMDP. Denoting the empty observation history at the first stage (i.e., $t = 0$) with the symbol $\vec{\underline{o}}^{-1}$, the value $V(\vec{\pi})$ of a joint policy equals $\sum_{s^0 \in S} \Delta_0(s^0) V^\pi(s^0, \vec{\underline{o}}^{-1})$.

DOI

7.2 Lifting SSAT to NEXPTIME-completeness

7.2.1 Formulation

In the following, we extend DQBF to its stochastic variant, named *dependency stochastic Boolean satisfiability* (DSSAT).

A DSSAT formula Φ over $V = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ is of the form:

$$\text{ᴙ}^{p_1} x_1, \ldots, \text{ᴙ}^{p_n} x_n, \exists y_1(D_{y_1}), \ldots, \exists y_m(D_{y_m}).\phi(x_1, \ldots, x_n, y_1, \ldots, y_m), \tag{7.3}$$

where each $D_{y_j} \subseteq \{x_1, \ldots, x_n\}$ denotes the set of variables that variable y_j depends on, and Boolean formula ϕ over V is quantifier-free. We denote the set $\{x_1, \ldots, x_n\}$ (resp. $\{y_1, \ldots, y_m\}$) of randomly (resp. existentially) quantified variables of Φ by $V_\Phi^{\text{ᴙ}}$ (resp. $V_\Phi^{\exists}$).

Given a DSSAT formula Φ and a set of Skolem functions $\mathcal{F} = \{f_j : \mathcal{A}(D_{y_j}) \mapsto \mathbb{B} \mid j = 1, \ldots, m\}$, the satisfying probability $\Pr[\Phi|_{\mathcal{F}}]$ of Φ with respect to $\mathcal{F}$ is defined as follows:

$$\Pr[\Phi|_{\mathcal{F}}] = \Pr[\text{ᴙ}^{p_1} x_1, \ldots, \text{ᴙ}^{p_n} x_n.\phi|_{\mathcal{F}}], \tag{7.4}$$

where $\phi|_{\mathcal{F}}$ denotes the formula obtained by substituting existentially quantified variables in ϕ with their respective Skolem functions, as defined in Section 7.1.1. In other words, the satisfying probability of Φ with respect to $\mathcal{F}$ equals the satisfying probability of the SSAT formula $\text{ᴙ}^{p_1} x_1, \ldots, \text{ᴙ}^{p_n} x_n.\phi|_{\mathcal{F}}$.

DOI

The *decision version* of DSSAT is stated as follows. Given a DSSAT formula Φ and a threshold $\theta \in [0, 1]$, decide whether there exists a set $\mathcal{F}$ of Skolem functions such that $\Pr[\Phi|_{\mathcal{F}}] \geq \theta$. On the other hand, the *optimization version* asks to find a set of Skolem functions to maximize the satisfying probability of Φ.

The formulation of SSAT can be extended by incorporating universal quantifiers, resulting in a unified framework named *extended* SSAT (XSSAT) [66], which subsumes both QBF and SSAT. In the extended SSAT, besides the four rules discussed in Section 3.3 to calculate the satisfying probability of an SSAT formula Φ, the following rule is added: $\Pr[\Phi] = \min\{\Pr[\Phi|_{\neg x}], \Pr[\Phi|_{x}]\}$, if x is universally quantified.

Similarly, an *extended* DSSAT (XDSSAT) formula Φ over a set of variables $\{x_1, \ldots, x_n, y_1, \ldots, y_m, z_1, \ldots, z_l\}$ is of the form:

$$Q_1 v_1, \ldots, Q_{n+l} v_{n+l}, \exists y_1(D_{y_1}), \ldots, \exists y_m(D_{y_m}).\phi, \tag{7.5}$$

where $Q_i v_i$ equals either $\text{ᴙ}^{p_k} x_k$ or $\forall z_k$ for some k with $v_i \neq v_j$ for $i \neq j$, and each $D_{y_j} \subseteq \{x_1, \ldots, x_n, z_1, \ldots, z_l\}$ denotes the set of randomly and universally quantified variables that variable y_j depends on.

The satisfying probability of an XDSSAT formula Φ with respect to a set of Skolem functions $\mathcal{F} = \{f_j : \mathcal{A}(D_{y_j}) \mapsto \mathbb{B} \mid j = 1, \ldots, m\}$, denoted by $\Pr[\Phi|_{\mathcal{F}}]$, equals the satisfying probability of the XSSAT formula $Q_1 v_1, \ldots, Q_{n+l} v_{n+l}.\phi|_{\mathcal{F}}$. This satisfiability definition subsumes those for DQBF (Eq. (7.1)) and DSSAT (Eq. (7.4)),

DOI

where the variables preceding the existential quantifiers in the prefixes are solely universally or randomly quantified.

Note that in the above extension the Henkin-type quantifiers are only defined for the existential variables. Although the extended formulation increases practical expressive succinctness, the computational complexity is not changed as to be shown in the next section.

7.2.2 Complexity proof

In the following, we show that the decision version of DSSAT is NEXPTIME-complete.

Theorem 7.1. *The decision version of DSSAT is NEXPTIME-complete.*

Proof. To show that DSSAT is NEXPTIME-complete, we have to show that it belongs to the NEXPTIME complexity class and that it is NEXPTIME-hard.

First, to see why DSSAT belongs to the NEXPTIME complexity class, observe that a Skolem function for an existentially quantified variable can be guessed and constructed in nondeterministic exponential time with respect to the number of randomly quantified variables. Given the guessed Skolem functions, the evaluation of the matrix, summation of weights of satisfying assignments, and comparison against the threshold θ can also be performed in exponential time. Overall, the

DOI

whole procedure is done in nondeterministic exponential time with respect to the input size, and hence DSSAT belongs to the NEXPTIME complexity class.

Second, to see why DSSAT is NEXPTIME-hard, we reduce the NEXPTIME-complete problem DQBF to DSSAT as follows. Given an arbitrary DQBF:

$$\Phi_Q = \forall x_1, \ldots, \forall x_n, \exists y_1(D_{y_1}), \ldots, \exists y_m(D_{y_m}).\phi,$$

we construct a DSSAT formula:

$$\Phi_S = \text{ꓤ}^{0.5} x_1, \ldots, \text{ꓤ}^{0.5} x_n, \exists y_1(D_{y_1}), \ldots, \exists y_m(D_{y_m}).\phi$$

by changing every universal quantifier to a randomized quantifier with probability 0.5. The reduction can be done in polynomial time with respect to the size of Φ_Q. We will show that Φ_Q is satisfiable if and only if there exists a set $\mathcal{F}$ of Skolem functions such that $\Pr[\Phi_S|_{\mathcal{F}}] \geq 1$.

The "only if" direction: As Φ_Q is satisfiable, there exists a set $\mathcal{F}$ of Skolem functions such that substituting the existentially quantified variables with the respective Skolem functions makes ϕ a tautology over variables $\{x_1, \ldots, x_n\}$. Therefore, $\Pr[\Phi_S|_{\mathcal{F}}] = \Pr[\text{ꓤ}^{0.5} x_1, \ldots, \text{ꓤ}^{0.5} x_n.\top] = 1 \geq 1$.

The "if" direction: As there exists a set $\mathcal{F}$ of Skolem functions such that $\Pr[\Phi_S|_{\mathcal{F}}] \geq 1$, after substituting the existentially quantified variables with the corresponding Skolem functions, each assignment $\tau \in \mathcal{A}(\{x_1, \ldots, x_n\})$ must satisfy ϕ, i.e., ϕ becomes a tautology over variables $\{x_1, \ldots, x_n\}$. Otherwise, the satisfying probability $\Pr[\Phi_S|_{\mathcal{F}}]$ must be less than 1 as the weight 2^{-n} of some unsatisfying

DOI

assignment is missing from the summation. Therefore, Φ_Q is satisfiable due to the existence of the set $\mathcal{F}$. ■

When DSSAT is extended with universal quantifiers, its complexity remains in the NEXPTIME complexity class as the fifth rule of the satisfying probability calculation does not incur any complexity overhead. Therefore the following corollary is immediate.

Corollary 7.1.1. *The decision problem of XDSSAT is NEXPTIME-complete.*

7.3 Applications of DSSAT

In this section, we demonstrate two applications of DSSAT.

7.3.1 Analyzing probabilistic/approximate partial design

After formulating DSSAT and proving its NEXPTIME-completeness, we show its application to the analysis of probabilistic design and approximate design. Specifically, we consider the probabilistic version of the *topologically constrained logic synthesis problem* [5, 100], or equivalently the *partial design problem* [36].

In the *(deterministic) partial design problem*, we are given a specification function $G(X)$ over primary input variables X and a *partial design* C_F with black boxes to

DOI

be synthesized. The Boolean functions induced at the primary outputs of C_F can be described by $F(X, T)$, where T corresponds to the variables of the black box outputs. Each black box output t_i is specified with its input variables (i.e., dependency set) $D_i \subseteq X \cup Y$ in C_F, where Y represents the variables for intermediate gates in C_F referred to by the black boxes. The partial design problem aims at deriving the black box functions $\{h_1(D_1), \ldots, h_{|T|}(D_{|T|})\}$ such that substituting t_i with h_i in C_F makes the resultant circuit function equal $G(X)$. The above partial design problem can be encoded as a DQBF problem; moreover, the partial equivalence checking problem is shown to be NEXPTIME-complete [36].

Specifically, the DQBF that encodes the partial equivalence checking problem is of the form:

$$\forall X, \forall Y, \exists T(D).(Y \equiv E(X)) \rightarrow (F(X, T) \equiv G(X)), \tag{7.6}$$

where D consists of $(D_1, \ldots, D_{|T|})$, E corresponds to the defining functions of Y in C_F, and the operator "$\equiv$" denotes element-wise equivalence between its two operands.

The above partial design problem can be extended to its probabilistic variant, which is illustrated by the circuit shown in Fig. 7.1. The *probabilistic partial design problem* is the same as the deterministic partial design problem except that C_F is a distilled probabilistic design [54] with black boxes, whose functions at the primary outputs can be described by $F(X, Z, T)$, where Z represents the variables for the auxiliary inputs that trigger errors in C_F (including the errors of the black boxes)

DOI

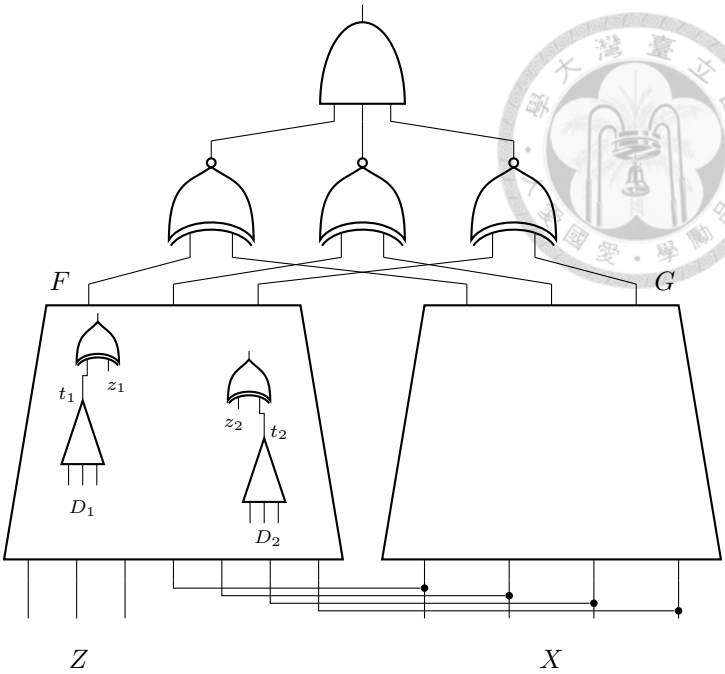


Figure 7.1: A miter for the equivalence checking of probabilistic partial design

and T corresponds to the variables of the black box outputs. Each black box output t_i is specified with its input variables (i.e., dependency set) $D_i \subseteq X \cup Y$ in C_F. When t_i is substituted with h_i in C_F, the function of the resultant circuit is required to be sufficiently close to the specification with respect to some expected probability.

Theorem 7.2. *The probabilistic partial design problem is NEXPTIME-complete.*

Proof. To show that the probabilistic partial design problem is in the NEXPTIME complexity class, we note that the black box functions can be guessed and validated in time exponential to the number of black box inputs.

To show completeness in the NEXPTIME complexity class, we reduce the known NEXPTIME-complete DSSAT problem to the probabilistic partial design problem,

DOI

similar to the construction used in the previous work [36]. Given a DSSAT instance, it can be reduced to a probabilistic partial design instance in polynomial time as follows. Without loss of generality, consider the DSSAT formula Eq. (7.3). We create a probabilistic partial design instance by letting the specification G be a tautology and letting C_F be a probabilistic design with black boxes, which involves primary inputs $x_1, \ldots, x_n$ and black box outputs $y_1, \ldots, y_m$ to compute the matrix ϕ. The driving inputs of the black box output y_j is specified by the dependency set D_{y_j} in Eq. (7.3), and the probability for primary input x_i to evaluate to $\top$ is set to p_i. The original DSSAT formula is satisfiable with respect to a target satisfying probability θ if and only if there exist implementations of the black boxes such that the resultant circuit composed with the black box implementations behaves like a tautology with respect to the required expectation θ. ■

On the other hand, the probabilistic partial design problem can be encoded with the following XDSSAT formula:

$$\aleph X, \aleph Z, \forall Y, \exists T(D).(Y \equiv E(X)) \rightarrow (F(X, Z, T) \equiv G(X)), \tag{7.7}$$

where the primary input variables are randomly quantified with probability p_{x_i} of $x_i \in X$ to reflect their weights, and the error-triggering auxiliary input variables Z are randomly quantified according to the pre-specified error rates of the erroneous gates in C_F. Notice that the above formula takes advantage of the extension with universal quantifiers as discussed previously.

In approximate design, a circuit implementation may deviate from its specifica-

DOI

tion by a certain extent. The amount of deviation can be characterized in a way similar to the error probability calculation in probabilistic design. For approximate partial design, the equivalence checking problem can be expressed by the XDSSAT formula:

$$\mathrm{Ƨ}X, \forall Y, \exists T(D).(Y \equiv E(X)) \rightarrow (F(X,T) \equiv G(X)), \tag{7.8}$$

which differs from Eq. (7.7) only in requiring no auxiliary inputs. The probabilities of the randomly quantified primary input variables are determined by the approximation criteria in measuring the deviation. For example, when all the input assignments are of equal weight, the probabilities of the primary inputs are all set to 0.5.

We note that as the engineering change order (ECO) problem [46] heavily relies on partial design equivalence checking, the above DSSAT formulations provide fundamental bases for ECOs of probabilistic and approximate designs.

7.3.2 Modeling Dec-POMDP

We demonstrate the descriptive power of DSSAT by constructing a polynomial-time reduction from Dec-POMDP to DSSAT. Our reduction is an extension of that from POMDP to SSAT proposed by Salmon and Poupart [94].

In essence, given a Dec-POMDP $\mathcal{M}$, we will construct in polynomial time a DSSAT formula Φ such that there is a joint policy $\vec{\pi}$ for $\mathcal{M}$ with value $V(\vec{\pi})$ if and

DOI

only if there is a set of Skolem functions $\mathcal{F}$ for Φ with satisfying probability $\Pr[\Phi|_{\mathcal{F}}]$, and $V(\vec{\pi}) = \Pr[\Phi|_{\mathcal{F}}]$.

First we introduce the variables used in construction of the DSSAT formula and their domains. To improve readability, we allow a variable x to take values from a finite set $U = \{x_1, \ldots, x_K\}$. Under this setting, a randomized quantifier ᴚ over variable x specifies a distribution $\Pr[x \equiv x_i]$ for each $x_i \in U$. We also define a scaled reward function:

$$r(s, \vec{a}) = \frac{\rho(s, \vec{a}) - \min\limits_{s', \vec{a}'} \rho(s', \vec{a}')}{\sum\limits_{s'', \vec{a}''} [\rho(s'', \vec{a}'') - \min\limits_{s', \vec{a}'} \rho(s', \vec{a}')]}$$

such that $r(s, \vec{a})$ forms a distribution over all pairs of s and $\vec{a}$, i.e., $\forall s, \vec{a}.r(s, \vec{a}) \geq 0$ and $\sum\limits_{s, \vec{a}} r(s, \vec{a}) = 1$. We will use the following variables:

- $x_s^t \in S$: the state at stage t,
- $x_a^{i,t} \in A_i$: the action taken by Agent i at stage t,
- $x_o^{i,t} \in O_i$: the observation received by Agent i at stage t,
- $x_r^t \in S \times (A_1 \times \ldots \times A_n)$: the reward earned at stage t,
- $x_T^t \in S$: transition distribution at stage t,
- $x_\Omega^t \in O_1 \times \ldots \times O_n$: observation distribution at stage t,
- $x_p^t \in \mathbb{B}$: used to sum up rewards across stages.

DOI

We encode elements in sets S, A_i, and O_i by integers, i.e., $S = \{0, 1, \ldots, |S|-1\}$, etc., and use indices s, a_i, and o_i to iterate through them, respectively. On the other hand, a special treatment is required for variables x_r^t and x_Ω^t, as they range over Cartesian products of multiple sets. We assign a unique number to an element in a product set as follows. Consider $\vec{Q} = Q_1 \times \ldots \times Q_n$, where each Q_i is a finite set. An element $\vec{q} = (q_1, \ldots, q_n) \in \vec{Q}$ is numbered by $N(q_1, \ldots, q_n) = \sum_{i=1}^{n} q_i(\prod_{j=1}^{i-1} |Q_j|)$. In the following construction, variables x_r^t and x_Ω^t will take values from the numbers given to the elements in $S \times \vec{A}$ and $\vec{O}$ by $N_r(s, \vec{a})$ and $N_\Omega(\vec{o})$, respectively.

We begin by constructing a DSSAT formula for a Dec-POMDP with $h = 1$. Under this setting, the derivation of an optimal joint policy is simply finding an action for each agent to maximize the expectation value of the reward function, i.e.,

$$\vec{a}^* = \arg\max_{\vec{a} \in \vec{A}} \sum_{s \in S} \Delta_0(s) r(s, \vec{a}).$$

The following DSSAT formula encodes the above optimization problem:

$$\text{ƀ} x_s^0, \text{ƀ} x_r^0, \exists x_a^{1,0}(D_{x_a^{1,0}}), \ldots, \exists x_a^{n,0}(D_{x_a^{n,0}}).\phi,$$

where the distribution of x_s^0 follows $\Pr[x_s^0 \equiv s] = \Delta_0(s)$, the distribution of x_r^0 follows $\Pr[x_r^0 \equiv N_r(s, \vec{a})] = r(s, \vec{a})$, each $D_{x_a^{i,0}} = \emptyset$, and the matrix:

$$\phi = \bigwedge_{s \in S} \bigwedge_{\vec{a} \in \vec{A}} [x_s^0 \equiv s \wedge \bigwedge_{i \in I} x_a^{i,0} \equiv a_i \rightarrow x_r^0 \equiv N_r(s, \vec{a})].$$

As the existentially quantified variables have no dependency on randomly quantified variable, the DSSAT formula is effectively an exist-random quantified SSAT formula.

DOI

For an arbitrary Dec-POMDP with $h > 1$, we follow the two steps proposed in the previous work [94], namely *policy selection* and *policy evaluation*, and adapt the policy selection step for the multi-agent setting in Dec-POMDP.

In the previous work [94], an agent's policy selection is encoded by the following prefix (use Agent i as an example):

$$\exists x_a^{i,0}, \text{ Ƨ} x_p^0, \text{ Ƨ} x_o^{i,0}, \ldots, \exists x_a^{i,h-2}, \text{ Ƨ} x_p^{h-2}, \text{ Ƨ} x_o^{i,h-2}, \exists x_a^{i,h-1}, \text{ Ƨ} x_p^{h-1}.$$

In the above quantification, variable x_p^t is introduced to sum up rewards earned at different stages. It takes values from $\mathbb{B}$, and follows a uniform distribution, i.e., $\Pr[x_p^t \equiv \top] = \Pr[x_p^t \equiv \bot] = 0.5$. When $x_p^t \equiv \bot$, the process is stopped and the reward at stage t is earned; when $x_p^t \equiv \top$, the process is continued to stage $t+1$. Note that variables $\{x_p^t\}$ are shared across all agents. With the help of variable x_p^t, rewards earned at different stages are summed up with an equal weight 2^{-h}. Variable $x_o^{i,t}$ also follows a uniform distribution $\Pr[x_o^{i,t} \equiv o_i] = |O_i|^{-1}$, which scales the satisfying probability by $|O_i|^{-1}$ at each stage. Therefore, we need to re-scale the satisfying probability accordingly in order to obtain the correct satisfying probability corresponding to the value of a joint policy. The scaling factor will be derived in the proof of Theorem 7.3.

As the actions of the agents can only depend on their own observation history, for the selection of a joint policy it is not obvious how to combine the quantification of each agent, i.e., the selection of an individual policy, into a linearly ordered prefix required by SSAT, without suffering an exponential translation cost. On the other

DOI

hand, DSSAT allows to specify the dependency of an existentially quantified variable freely and is suitable to encode the selection of a joint policy. In the prefix of the DSSAT formula, variable $x_a^{i,t}$ depends on $D_{x_a^{i,t}} = \{x_o^{i,0}, \ldots, x_o^{i,t-1}, x_p^0, \ldots, x_p^{t-1}\}$.

Next, the policy evaluation step is exactly the same as that in the previous work [94]. The following quantification computes the value of a joint policy:

$$\exists x_s^t, \exists x_r^t, t = 0, \ldots, h-1$$

$$\exists x_T^t, \exists x_\Omega^t, t = 0, \ldots, h-2$$

Variables x_s^t follow a uniform distribution $\Pr[x_s^t \equiv s] = |S|^{-1}$ except for variable x_s^0, which follows the initial distribution specified by $\Pr[x_s^0 \equiv s] = \Delta_0(s)$; variables x_r^t follow the distribution of the reward function $\Pr[x_r^t \equiv N_r(s, \vec{a})] = r(s, \vec{a})$; variables x_T^t follow the state transition distribution $\Pr[x_{T_{s,\vec{a}}}^t \equiv s'] = T(s, \vec{a}, s')$; variables x_Ω^t follow the observation distribution $\Pr[x_{\Omega_{s',\vec{a}}}^t \equiv N_\Omega(\vec{o})] = \Omega(s', \vec{a}, \vec{o})$. Note that these variables encode the random mechanism of a Dec-POMDP and are hidden from the agents. That is, variables $x_a^{i,t}$ do not depend on the above variables.

The formulas to encode $\mathcal{M}$ are listed in Fig. 7.2. Eq. (7.9) encodes that when $x_p^t \equiv \bot$, i.e., the process is stopped, the observation $x_o^{i,t}$ and next state x_s^{t+1} are set to a preserved value 0, and $x_p^{t+1} \equiv \bot$. Eq. (7.10) ensures the process is stopped at the last stage. Eq. (7.11) ensures the reward at the first stage is earned when the process is stopped, i.e., $x_p^0 \equiv \bot$. Eq. (7.12) requires the reward at stage $t > 0$ is earned when $x_p^{t-1} \equiv \top$ and $x_p^t \equiv \bot$. Eq. (7.13) encodes the transition distribution from state s to state s' given actions $\vec{a}$ are taken. Eq. (7.14) encodes the observation

DOI

$$\bigwedge_{0\le t\le h-2} [x_p^t \equiv \bot \rightarrow \bigwedge_{i\in I} x_o^{i,t} \equiv 0 \wedge x_s^{t+1} \equiv 0 \wedge x_p^{t+1} \equiv \bot] \tag{7.9}$$

$$x_p^{h-1} \equiv \bot \tag{7.10}$$

$$\bigwedge_{s\in S} \bigwedge_{\vec{a}\in\vec{A}} [x_p^0 \equiv \bot \wedge x_s^0 \equiv s \wedge \bigwedge_{i\in I} x_a^{i,0} \equiv a_i \rightarrow x_r^0 \equiv N_r(s,\vec{a})] \tag{7.11}$$

$$\bigwedge_{1\le t\le h-1} \bigwedge_{s\in S} \bigwedge_{\vec{a}\in\vec{A}} [x_p^{t-1} \equiv \top \wedge x_p^t \equiv \bot \wedge x_s^t \equiv s \wedge \bigwedge_{i\in I} x_a^{i,t} \equiv a_i \rightarrow x_r^t \equiv N_r(s,\vec{a})] \tag{7.12}$$

$$\bigwedge_{0\le t\le h-2} \bigwedge_{s\in S} \bigwedge_{\vec{a}\in\vec{A}} \bigwedge_{s'\in S} [x_p^t \equiv \top \wedge x_s^t \equiv s \wedge \bigwedge_{i\in I} x_a^{i,t} \equiv a_i \wedge x_s^{t+1} \equiv s' \rightarrow x_{T_{s,\vec{a}}}^t \equiv s'] \tag{7.13}$$

$$\bigwedge_{0\le t\le h-2} \bigwedge_{s'\in S} \bigwedge_{\vec{a}\in\vec{A}} \bigwedge_{\vec{o}\in\vec{O}} [x_p^t \equiv \top \wedge x_s^{t+1} \equiv s' \wedge \bigwedge_{i\in I} x_a^{i,t} \equiv a_i \wedge \bigwedge_{i\in I} x_o^{i,t} \equiv o_i \rightarrow x_{\Omega_{s',\vec{a}}}^t \equiv N_\Omega(\vec{o})] \tag{7.14}$$

Figure 7.2: The formulas used to encode a Dec-POMDP $\mathcal{M}$

distribution to receive observation $\vec{o}$ under the situation that state s' is reached after actions $\vec{a}$ are taken.

The following theorem states the correctness of the reduction.

Theorem 7.3. *The above reduction maps a Dec-POMDP $\mathcal{M}$ to a DSSAT formula Φ, such that a joint policy $\vec{\pi}$ exists for $\mathcal{M}$ if and only if a set of Skolem functions $\mathcal{F}$ exists for Φ, with $V(\vec{\pi}) = \Pr[\Phi|_{\mathcal{F}}]$.*

Proof. Given an arbitrary Dec-POMDP $\mathcal{M}$, we prove the statement via mathematical induction over the planning horizon h as follows.

For the base case $h = 1$, to prove the "only if" direction, consider a joint policy $\vec{\pi}$ for $\mathcal{M}$ that specifies $\vec{a} = (a_1, \ldots, a_n)$ where Agent i takes action a_i. For this joint policy, the value is computed as $V(\vec{\pi}) = \sum_{s\in S} \Delta_0(s) r(s,\vec{a})$. Based on $\vec{\pi}$, we

DOI

$$
\begin{aligned}
\Pr[\Phi|_{\mathcal{F}_{h+1}}] &= \sum_{v^0,\ldots,v^h}^{2} \sum_{\vec{o}^0,\ldots,\vec{o}^{h-1}}^{|\vec{O}|} \sum_{s^0,\ldots,s^h}^{|S|} \prod_{t=0}^{h} \Pr[x_p^t \equiv v^t, x_s^t \equiv s^t, \vec{x_o^t} \equiv \vec{o}^t, x_r^t] \prod_{t=0}^{h-1} \Pr[x_T^t, x_\Omega^t | v^t, s^t, \vec{o}^t] \\
&= 2^{-(h+1)} \sum_{\hat{t}=1}^{h+1} \sum_{\vec{o}^0,\ldots,\vec{o}^{\hat{t}-2}}^{|\vec{O}|} \sum_{s^0,\ldots,s^{\hat{t}-1}}^{|S|} \prod_{t=0}^{\hat{t}-1} \Pr[x_s^t \equiv s^t, \vec{x_o^t} \equiv \vec{o}^t, x_r^t] \prod_{t=0}^{\hat{t}-2} \Pr[x_T^t, x_\Omega^t | v^t, s^t, \vec{o}^t] \\
&= 2^{-(h+1)} |\vec{O}|^{-h} \sum_{\hat{t}=1}^{h+1} \sum_{\vec{o}^0,\ldots,\vec{o}^{\hat{t}-2}}^{|\vec{O}|} \sum_{s^0,\ldots,s^{\hat{t}-1}}^{|S|} \prod_{t=0}^{\hat{t}-1} \Pr[x_s^t \equiv s^t, x_r^t] \prod_{t=0}^{\hat{t}-2} \Pr[x_T^t, x_\Omega^t | v^t, s^t, \vec{o}^t] \\
&= 2^{-(h+1)} (|\vec{O}| \cdot |S|)^{-h} \sum_{\hat{t}=1}^{h+1} \sum_{\vec{o}^0,\ldots,\vec{o}^{\hat{t}-2}}^{|\vec{O}|} \sum_{s^0,\ldots,s^{\hat{t}-1}}^{|S|} \Pr[x_s^0 \equiv s^0] \prod_{t=0}^{\hat{t}-1} \Pr[x_r^t] \prod_{t=0}^{\hat{t}-2} \Pr[x_T^t, x_\Omega^t | v^t, s^t, \vec{o}^t] \\
&= 2^{-(h+1)} (|\vec{O}| \cdot |S|)^{-h} \sum_{\hat{t}=1}^{h+1} \sum_{\vec{o}^0,\ldots,\vec{o}^{\hat{t}-2}}^{|\vec{O}|} \sum_{s^0,\ldots,s^{\hat{t}-1}}^{|S|} \Pr[x_s^0 \equiv s^0] \Pr[x_r^{\hat{t}-1}] \prod_{t=0}^{\hat{t}-2} \Pr[x_T^t, x_\Omega^t | v^t, s^t, \vec{o}^t] \\
&= \kappa_{h+1}^{-1} \sum_{\hat{t}=1}^{h+1} \sum_{\vec{o}^0,\ldots,\vec{o}^{\hat{t}-2}}^{|\vec{O}|} \sum_{s^0,\ldots,s^{\hat{t}-1}}^{|S|} \Delta_0(s^0) r(s^{\hat{t}-1}, \vec{a}^{\hat{t}-1}) \prod_{t=0}^{\hat{t}-2} T(s^t, \vec{a}^t, s^{t+1}) \Omega(s^{t+1}, \vec{a}^t, \vec{o}^t) \\
&= \kappa_{h+1}^{-1} \sum_{s^0 \in S} \Delta_0(s^0)(r(s^0, \vec{a}^0) + \sum_{\vec{o}^0 \in \vec{O}} \sum_{s^1 \in S} T(s^0, \vec{a}^0, s^1) \Omega(s^1, \vec{a}^0, \vec{o}^0) \Pr[\Phi|_{\mathcal{F}_h}]) \\
&= \kappa_{h+1}^{-1} \sum_{s^0 \in S} \Delta_0(s^0)(r(s^0, \vec{a}^0) + \sum_{\vec{o}^0 \in \vec{O}} \sum_{s^1 \in S} T(s^0, \vec{a}^0, s^1) \Omega(s^1, \vec{a}^0, \vec{o}^0) V(\vec{\pi}_h)) \text{(induction hypothesis)} \\
&= \kappa_{h+1}^{-1} V(\vec{\pi}_{h+1}) \text{(by Eq. (7.2))}
\end{aligned}
$$

Figure 7.3: The derivation of the induction case in the proof of Theorem 7.3

construct a set of Skolem functions $\mathcal{F}$ where $x_a^{i,0} = a_i$ for each $i \in I$. To compute $\Pr[\Phi|_{\mathcal{F}}]$, we cofactor the matrix with $\mathcal{F}$ and arrive at the following CNF formula:

$$\bigwedge_{s \in S} [x_s^0 \neq s \vee x_r^0 \equiv N_r(s, \vec{a})],$$

and the satisfying probability of Φ with respect to $\mathcal{F}$ is

$$\Pr[\Phi|_{\mathcal{F}}] = \sum_{s \in S} \Pr[x_s^0 \equiv s] \Pr[x_r^0 \equiv N_r(s, \vec{a})] = \sum_{s \in S} \Delta_0(s) r(s, \vec{a}) = V(\vec{\pi}).$$

Note that only equalities are involved in the above argument. The reasoning steps can hence be reversed to prove the "if" direction.

DOI

For the induction step, first assume that the statement holds for a planning horizon $h > 1$. For a planning horizon of $h + 1$, consider a joint policy $\vec{\pi}_{h+1}$ with value $V(\vec{\pi}_{h+1})$. Note that as a joint policy is a mapping from observation histories to actions, we can build a corresponding set of Skolem functions $\mathcal{F}_{h+1}$ to simulate joint policy $\vec{\pi}_{h+1}$ for the DSSAT formula. The derivation of satisfying probability with respect to $\mathcal{F}_{h+1}$ is shown in Fig. 7.3. Note that to obtain the correct value of the joint policy, we need to re-scale the satisfying probability by a scaling factor $\kappa_{h+1} = 2^{h+1}(|\vec{O}||S|)^h$. As only equalities are involved in the derivation in Fig. 7.3, the "if" direction is also proved.

Because $\Pr[\Phi|_{\mathcal{F}_{h+1}}] = V(\vec{\pi}_{h+1})$ is established, the theorem is proved according to the principle of mathematical induction. ■

Discussion

Below we count the numbers of variables and clauses in the resulting DSSAT formula with respect to the input size of the given Dec-POMDP. For a stage, there are $3+2(|I|+|S||\vec{A}|)$ variables, and therefore in total the number of variables is $O(h(|I|+|S||A|))$ asymptotically. On the other hand, the number of clauses per stage is $2+|I|+|S||\vec{A}|+|S|^2|\vec{A}|+|S||\vec{A}||\vec{O}|$, and hence the total number of clauses is $O(h(|I|+|S||\vec{A}|(|S|+|\vec{O}|))$. Overall, we show that the proposed reduction is polynomial-time with respect to the input size of the Dec-POMDP.

DOI

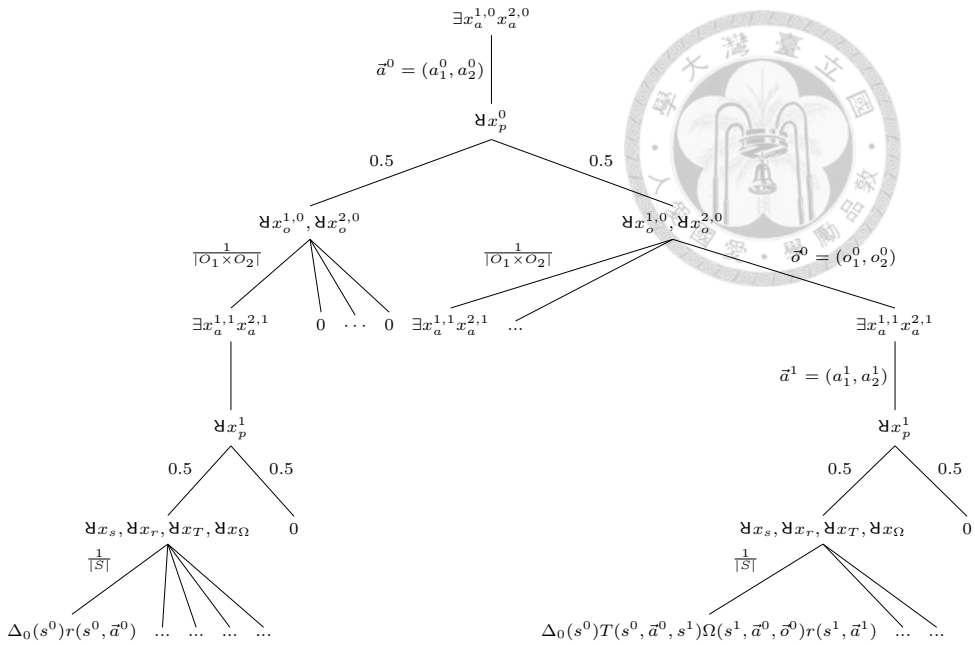


Figure 7.4: A Dec-POMDP example with two agents and $h = 2$

Below we demonstrate the reduction with an example.

Example 7.1. *Consider a Dec-POMDP with two agents and planning horizon $h = 2$. Given a joint policy (π_1, π_2) for Agent 1 and Agent 2, let the actions taken at $t = 0$ be $\vec{a}^0 = (a_1^0, a_2^0)$ and the actions taken at $t = 1$ under observations $\vec{o}^0 = (o_1^0, o_2^0)$ be $\vec{a}^1 = (a_1^1, a_2^1)$. The value of this joint policy is computed by Eq. (7.2) as:*

$$V(\pi) = \sum_{s^0 \in S} \Delta_0(s^0)[r(s^0, \vec{a}^0) + \sum_{\vec{o}^0 \in \vec{O}} \sum_{s^1 \in S} T(s^0, \vec{a}^0, s^1)\Omega(s^1, \vec{a}^0, \vec{o}^0) r(s^1, \vec{a}^1)].$$

The decision tree of the converted DSSAT formula is shown in Fig. 7.4. At $t = 0$, after taking actions $\vec{a}^0$, variable x_p^0 splits into two cases: when $x_p^0 \equiv \bot$ (left branch), the expected reward $\Delta_0(s^0) r(s^0, \vec{a}^0)$ will be earned for $t = 0$; on the other hand, when $x_p^0 \equiv \top$ (right branch), observation $\vec{o}^0$ is received, based on which the agents will select their actions $\vec{a}^1$ at $t = 1$. Again, variable x_p^1 will split into two

DOI

cases, but this time x_p^1 *is forced to be* $\perp$ *as it is the last stage. The expected reward* $\Delta_0(s^0)T(s^0, \vec{a}^0, s^1)\Omega(s^1, \vec{a}^0, \vec{o}^0)r(s^1, \vec{a}^1)$ *will be earned under the branch of* $x_p^1 \equiv \perp$ *for* $t = 1$*. Note that the randomized quantifiers over variables* x_p^t*,* x_s^t*, and* x_o^t *will scale the satisfying probability by the factors labelled on the edges, respectively. Therefore, we have to re-scale the satisfying probability by* $2^2|S||O_1 \times O_2|$*, which is predicted by the scaling factor* $\kappa_h = 2^h(|\vec{O}||S|)^{h-1}$ *calculated in the proof of Theorem 7.3.*

DOI

Chapter 8

Conclusion and Future Work

In this dissertation, the research needs highlighted in Chapter 1 have been addressed. Specifically, the applicability of SSAT to the analysis of VLSI systems was examined. We formulated a framework for the property evaluation of probabilistic design, and encoded the average-case and the worst-case analyses as random-exist and exist-random quantified SSAT formulas, respectively.

Motivated by the emerging VLSI applications, we further designed novel algorithms for random-exist and exist-random quantified SSAT formulas. The proposed algorithms aim at leveraging the success from SAT/QBF-solving and model-counting communities, so as to advance the state-of-the-art of SSAT solving beyond the conventional DPLL-based search. For random-exist quantified SSAT formulas, we used minterm generalization and weighted model counting as subroutines, and employed

DOI

SAT solvers and weighted model counters as plug-in engines. For exist-random quantified SSAT formulas, we devised the clause-containment learning, which was inspired by the clause-selection technique of QBF. Under the framework of clause-containment learning, three enhancement techniques were proposed to improve the computational efficiency. Moreover, unlike previous exact SSAT methods, the proposed algorithms can solve approximate SSAT by deriving upper and lower bounds of satisfying probability. Experiment results showed the benefits of our solvers over a wide range of benchmarking instances. Our implementations and the benchmark suite of SSAT instances are open-source for other researchers to base their work on top of our results.

To generalize SSAT beyond the PSPACE-complete complexity class for more complex problems, we extended DQBF to its stochastic variant DSSAT and proved its NEXPTIME-completeness. Compared to the PSPACE-complete SSAT, DSSAT is more powerful to succinctly model NEXPTIME-complete decision problems with uncertainty. We demonstrated the DSSAT formulation of the analysis to probabilistic/approximate partial design, and gave a polynomial-time reduction from the NEXPTIME-complete Dec-POMDP to DSSAT.

For future work, we plan to incorporate Monte-Carlo simulation into our framework of probabilistic property evaluation, and develop solvers for arbitrary quantified SSAT and DSSAT formulas. We note that the clause-containment learning has been generalized to solve general SSAT [19], and the recent developments of DQBF [104]

DOI

might provide a promising framework for DSSAT solving.

DOI

List of Figures

4.1 Distillation of a NAND gate with an error rate p 28

4.2 A miter SPBN for probabilistic property evaluation 31

5.1 Quantile plots of random k-CNF formulas 79

5.2 Quantile plots of strategic-company formulas 81

5.3 Run-time scatter plots of strategic-company formulas with `reSSAT` in y-axis and compared approaches in x-axis 82

6.1 Quantile plots of random k-CNF formulas 107

6.2 Quantile plots of application formulas 110

6.3 Run-time scatter plots of application formulas with `erSSAT` in y-axis and compared approaches in x-axis 111

DOI

7.1 A miter for the equivalence checking of probabilistic partial design . . 129

7.2 The formulas used to encode a Dec-POMDP $\mathcal{M}$ 136

7.3 The derivation of the induction case in the proof of Theorem 7.3 . . . 137

7.4 A Dec-POMDP example with two agents and $h = 2$ 139

DOI

List of Tables

2.1 Model-counting variants and their corresponding SSAT formulas . . . 15

3.1 Summary of the symbols used in the dissertation 23

4.1 Circuit statistics of ISCAS benchmark suite 51

4.2 Circuit statistics of EPFL benchmark suite 52

4.3 Miter statistics of ISCAS benchmark suite ($\delta = 0.01$) 53

4.4 Miter statistics of ISCAS benchmark suite ($\delta = 0.1$) 54

4.5 Miter statistics of EPFL benchmark suite ($\delta = 0.01$) 55

4.6 Miter statistics of EPFL benchmark suite ($\delta = 0.1$) 56

4.7 Solving PEC by various techniques ($\delta = 0.01$) 58

4.8 Solving PEC by various techniques ($\delta = 0.1$) 59

DOI

4.9 Solving MPEC by various techniques ($\delta = 0.01$) 60

4.10 Solving MPEC by various techniques ($\delta = 0.1$) 61

5.1 Summary of the results for 60 strategic-company formulas 80

5.2 Summary of the results for 60 PEC formulas 83

5.3 Results of solving the PEC formulas ($\delta = 0.01$) 84

5.4 Results of solving the PEC formulas ($\delta = 0.1$) 85

6.1 The families of the application formulas 105

6.2 Summary of the results for 212 application formulas 108

6.3 Results of solving the *Conformant* family 116

6.4 Results of solving the *Max-Count* family 117

6.5 Results of solving the *MPEC* family 118

DOI

List of Algorithms

1 BDD-based SSAT solving: `BddSsatSolve` 35

2 The recursive step of `BddSsatSolve`: `BddSsatRecur` 36

3 Formula rewriting for unweighted model counting: `WmcRewriting` . . 42

4 Solving random-exist quantified SSAT formulas 67

5 Solving exist-random quantified SSAT (E-MAJSAT) formulas 91

6 Subroutine of Alg. 5: `SelectMinimalClauses` 96

7 Subroutine of Alg. 5: `RemoveSubsumedClauses` 98

8 Subroutine of Alg. 5: `DiscardLiterals` 101

DOI

Bibliography

[1] L. Amarú, P.-E. Gaillardon, and G. De Micheli. 2015. The EPFL combinational benchmark suite. In *Proc. IWLS*. https://www.epfl.ch/labs/lsi/page-102566-en-html/benchmarks/

[2] R. A. Aziz, G. Chu, C. J. Muise, and P. J. Stuckey. 2015. #∃SAT: Projected model counting. In *Proc. SAT (LNCS 9340)*. Springer, 121–137. https://doi.org/10.1007/978-3-319-24318-4_10

[3] F. Bacchus, S. Dalmao, and T. Pitassi. 2003. Algorithms and complexity results for #SAT and Bayesian inference. In *Proc. FOCS*. IEEE Computer Society, 340–351. https://doi.org/10.1109/SFCS.2003.1238208

[4] R. I. Bahar, J. L. Mundy, and J. Chen. 2003. A probabilistic-based design methodology for nanoscale computation. In *Proc. ICCAD*. IEEE Computer Society / ACM, 480–486. https://doi.org/10.1109/ICCAD.2003.1257854

[5] V. Balabanov, H.-J. K. Chiang, and J.-H. R. Jiang. 2014. Henkin quantifiers and Boolean formulae: A certification perspective of DQBF. *Theoretical Com-*

DOI

puter Science 523 (2014), 86–100. https://doi.org/10.1016/j.tcs.2013.12.020

[6] C. W. Barrett and C. Tinelli. 2018. Satisfiability modulo theories. In *Handbook of Model Checking*. Springer, 305–343. https://doi.org/10.1007/978-3-319-10575-8_11

[7] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen. 2013. *Systems and Software Verification: Model-Checking Techniques and Tools* (1st ed.). Springer. https://doi.org/10.1007/978-3-662-04558-9

[8] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research* 27, 4 (2002), 819–840. https://doi.org/10.1287/moor.27.4.819.297

[9] D. Beyer, S. Löwe, and P. Wendler. 2019. Reliable benchmarking: Requirements and solutions. *International Journal on Software Tools for Technology Transfer* 21, 1 (2019), 1–29. https://doi.org/10.1007/s10009-017-0469-y

[10] A. Biere, M. Heule, H. van Maaren, and T. Walsh (Eds.). 2009. *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press.

[11] R. K. Brayton and A. Mishchenko. 2010. ABC: An academic industrial-strength verification tool. In *Proc. CAV (LNCS 6174)*. Springer, 24–40. https://doi.org/10.1007/978-3-642-14295-6_5 Latest version available at: https://github.com/berkeley-abc/abc.

[12] F. Brglez and H. Fujiwara. 1985. A neutral netlist of 10 combinational benchmark circuits. In *Proc. ISCAS*. IEEE, 695–698. https://people.engr.ncsu.edu/brglez/CBL/benchmarks/index.html

[13] H. K. Büning and U. Bubeck. 2009. Theory of quantified Boolean formulas. In *Handbook of Satisfiability*. IOS Press, 735–760. https://doi.org/10.3233/978-1-58603-929-5-735

[14] M. Cadoli, T. Eiter, and G. Gottlob. 1997. Default logic as a query language. *IEEE Transactions on Knowledge and Data Engineering* 9, 3 (1997), 448–463. https://doi.org/10.1109/69.599933

[15] S. Chakraborty, K. S. Meel, and M. Y. Vardi. 2013. A scalable approximate model counter. In *Proc. CP (LNCS 8124)*. Springer, 200–216. https://doi.org/10.1007/978-3-642-40627-0_18

[16] S. Chakraborty, K. S. Meel, and M. Y. Vardi. 2016. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proc. IJCAI*. IJCAI/AAAI Press, 3569–3576. http://www.ijcai.org/Abstract/16/503

[17] L. N. B. Chakrapani, J. George, B. Marr, B. E. S. Akgul, and K. V. Palem. 2006. Probabilistic design: A survey of probabilistic CMOS technology and future directions for terascale IC design. In *Proc. VLSI-SoC (IFIP 249)*. Springer, 101–118. https://doi.org/10.1007/978-0-387-74909-9_7

[18] M. Chavira and A. Darwiche. 2008. On probabilistic inference by weighted model counting. *Artificial Intelligence* 172, 6-7 (2008), 772–799. https://doi.org/10.1016/j.artint.2007.11.002

[19] P.-W. Chen, Y.-C. Huang, and J.-H. R. Jiang. 2021. A sharp leap from quantified Boolean formula to stochastic Boolean satisfiability solving. In *Proc. AAAI*. AAAI Press.

[20] M. R. Choudhury and K. Mohanram. 2009. Reliability analysis of logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 3 (2009), 392–405. https://doi.org/10.1109/TCAD.2009.2012530

[21] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu. 2001. Bounded model checking using satisfiability solving. *Formal Methods in System Design* 19, 1 (2001), 7–34. https://doi.org/10.1023/A:1011276507260

[22] C. Constantinescu. 2003. Trends and challenges in VLSI circuit reliability. *IEEE Micro* 23, 4 (2003), 14–19. https://doi.org/10.1109/MM.2003.1225959

[23] S. A. Cook. 1971. The complexity of theorem-proving procedures. In *Proc. STOC*. ACM, 151–158. https://doi.org/10.1145/800157.805047

[24] G. F. Cooper. 1990. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence* 42, 2-3 (1990), 393–405. https://doi.org/10.1016/0004-3702(90)90060-D

[25] A. Darwiche. 2001. Decomposable negation normal form. *J. ACM* 48, 4 (2001), 608–647. https://doi.org/10.1145/502090.502091

[26] A. Darwiche. 2002. A compiler for deterministic, decomposable negation normal form. In *Proc. AAAI*. AAAI Press / The MIT Press, 627–634. http://www.aaai.org/Library/AAAI/2002/aaai02-094.php

[27] A. Darwiche and P. Marquis. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17 (2002), 229–264. https://doi.org/10.1613/jair.989

[28] M. Davis, G. Logemann, and D. W. Loveland. 1962. A machine program for theorem-proving. *Commun. ACM* 5, 7 (1962), 394–397. https://doi.org/10.1145/368273.368557

[29] L. M. de Moura and N. Bjørner. 2011. Satisfiability modulo theories: Introduction and applications. *Commun. ACM* 54, 9 (2011), 69–77. https://doi.org/10.1145/1995376.1995394

[30] R. Dechter. 1998. Bucket elimination: A unifying framework for probabilistic inference. In *Learning in Graphical Models*. NATO ASI Series, Vol. 89. Springer, 75–104. https://doi.org/10.1007/978-94-011-5014-9_4

[31] N. Eén and N. Sörensson. 2003. An extensible SAT-solver. In *Proc. SAT (LNCS 2919)*. Springer, 502–518. https://doi.org/10.1007/978-3-540-24605-3_37

[32] N. Eén and N. Sörensson. 2003. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science* 89, 4 (2003), 543–560. https://doi.org/10.1016/S1571-0661(05)82542-3

[33] W. Faber and F. Ricca. 2005. Solving hard ASP programs efficiently. In *Proc. LPNMR (LNCS 3662)*. Springer, 240–252. https://doi.org/10.1007/11546207_19

[34] J. K. Fichte, M. Hecher, and F. Hamiti. 2020. The Model Counting Competition 2020. *CoRR* abs/2012.01323 (2020). arXiv:2012.01323 https://arxiv.org/abs/2012.01323

[35] D. J. Fremont, M. N. Rabe, and S. A. Seshia. 2017. Maximum model counting. In *Proc. AAAI*. AAAI Press, 3885–3892. http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14968

[36] K. Gitina, S. Reimer, M. Sauer, R. Wimmer, C. Scholl, and B. Becker. 2013. Equivalence checking of partial designs using dependency quantified Boolean

formulae. In *Proc. ICCD*. 396–403. https://doi.org/10.1109/ICCD.2013.6657071

[37] C. P. Gomes, J. Hoffmann, A. Sabharwal, and B. Selman. 2007. From sampling to model counting. In *Proc. IJCAI*. Morgan Kaufmann, 2293–2299. http://ijcai.org/Proceedings/07/Papers/369.pdf

[38] C. P. Gomes, A. Sabharwal, and B. Selman. 2006. Model counting: A new strategy for obtaining good bounds. In *Proc. AAAI*. AAAI Press, 54–61. http://www.aaai.org/Library/AAAI/2006/aaai06-009.php

[39] C. P. Gomes, A. Sabharwal, and B. Selman. 2009. Model counting. In *Handbook of Satisfiability*. IOS Press, 633–654. https://doi.org/10.3233/978-1-58603-929-5-633

[40] H. Hansson and B. Jonsson. 1989. A framework for reasoning about time and reliability. In *Proc. RTSS*. IEEE Computer Society, 102–111. https://doi.org/10.1109/REAL.1989.63561

[41] B. Hnich, R. Rossi, S. A. Tarim, and S. Prestwich. 2011. A survey on CP-AI-OR hybrids for decision making under uncertainty. In *Hybrid Optimization: The Ten Years of CPAIOR*. Springer, 227–270. https://doi.org/10.1007/978-1-4419-1644-0_7

[42] J. Huang. 2006. Combining knowledge compilation and search for conformant

probabilistic planning. In *Proc. ICAPS*. AAAI, 253–262. http://www.aaai.org/Library/ICAPS/2006/icaps06-026.php

[43] M. Janota and J. P. Marques-Silva. 2015. Solving QBF by clause selection. In *Proc. IJCAI*. AAAI Press, 325–331. http://ijcai.org/Abstract/15/052

[44] F. V. Jensen. 1996. *An Introduction to Bayesian Networks*. Taylor & Francis.

[45] R. Jhala and R. Majumdar. 2009. Software model checking. *Comput. Surveys* 41, 4 (2009), 21:1–21:54. https://doi.org/10.1145/1592434.1592438

[46] J.-H. R. Jiang, V. N. Kravets, and N.-Z. Lee. 2020. Engineering change order for combinational and sequential design rectification. In *Proc. DATE*. IEEE, 726–731. https://doi.org/10.23919/DATE48585.2020.9116504

[47] A. B. Kahng and S. Kang. 2012. Accuracy-configurable adder for approximate arithmetic designs. In *Proc. DAC*. ACM, 820–825. https://doi.org/10.1145/2228360.2228509

[48] Y. Kim, Y. Zhang, and P. Li. 2013. An energy efficient approximate adder with carry skip for error resilient neuromorphic VLSI systems. In *Proc. ICCAD*. IEEE, 130–137. https://doi.org/10.1109/ICCAD.2013.6691108

[49] S. Krishnaswamy, G. F. Viamontes, I. L. Markov, and J. P. Hayes. 2005. Accurate reliability evaluation and enhancement via probabilistic transfer matrices. In *Proc. DATE*. IEEE Computer Society, 282–287. https://doi.org/10.1109/DATE.2005.47

[50] A. Kuehlmann and F. Krohm. 1997. Equivalence checking using cuts and heaps. In *Proc. DAC*. ACM Press, 263–268. https://doi.org/10.1145/266021.266090

[51] N. Kushmerick, S. Hanks, and D. S. Weld. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76, 1-2 (1995), 239–286. https://doi.org/10.1016/0004-3702(94)00087-H

[52] M. Z. Kwiatkowska, G. Norman, and D. Parker. 2002. PRISM: Probabilistic symbolic model checker. In *Proc. TOOLS (LNCS 2324)*. Springer, 200–204. https://doi.org/10.1007/3-540-46029-2_13

[53] M. Lauria, J. Elffers, J. Nordström, and M. Vinyals. 2017. CNFgen: A generator of crafted benchmarks. In *Proc. SAT (LNCS 10491)*. Springer, 464–473. https://doi.org/10.1007/978-3-319-66263-3_30

[54] N.-Z. Lee and J.-H. R. Jiang. 2018. Towards formal evaluation and verification of probabilistic design. *IEEE Trans. Comput.* 67, 8 (2018), 1202–1216. https://doi.org/10.1109/TC.2018.2807431

[55] N.-Z. Lee and J.-H. R. Jiang. 2021. Dependency stochastic Boolean satisfiability: A logical formalism for NEXPTIME decision problems with uncertainty. In *Proc. AAAI*. AAAI Press.

[56] N.-Z. Lee, Y.-S. Wang, and J.-H. R. Jiang. 2017. Solving stochastic Boolean

satisfiability under random-exist quantification. In *Proc. IJCAI*. IJCAI Organization, 688–694. https://doi.org/10.24963/ijcai.2017/96

[57] N.-Z. Lee, Y.-S. Wang, and J.-H. R. Jiang. 2018. Solving exist-random quantified stochastic Boolean satisfiability via clause selection. In *Proc. IJCAI*. IJCAI Organization, 1339–1345. https://doi.org/10.24963/ijcai.2018/186

[58] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic* 7, 3 (2006), 499–562. https://doi.org/10.1145/1149114.1149117

[59] L. Li and H. Zhou. 2014. On error modeling and analysis of approximate adders. In *Proc. ICCAD*. IEEE, 511–518. https://doi.org/10.1109/ICCAD.2014.7001399

[60] K. Lingasubramanian, S. M. Alam, and S. Bhanja. 2011. Maximum error modeling for fault-tolerant computation using maximum a posteriori (MAP) hypothesis. *Microelectronics Reliability* 51, 2 (2011), 485–501. https://doi.org/10.1016/j.microrel.2010.07.156

[61] K. Lingasubramanian and S. Bhanja. 2007. Probabilistic maximum error modeling for unreliable logic circuits. In *Proc. GLSVLSI*. ACM, 223–226. https://doi.org/10.1145/1228784.1228842

[62] M. L. Littman, J. Goldsmith, and M. Mundhenk. 1998. The computational

complexity of probabilistic planning. *Journal of Artificial Intelligence Research* 9 (1998), 1–36. https://doi.org/10.1613/jair.505

[63] M. L. Littman, S. M. Majercik, and T. Pitassi. 2001. Stochastic Boolean satisfiability. *Journal of Automated Reasoning* 27, 3 (2001), 251–296. https://doi.org/10.1023/A:1017584715408

[64] S. M. Majercik. 2004. Nonchronological backtracking in stochastic Boolean satisfiability. In *Proc. ICTAI*. IEEE Computer Society, 498–507. https://doi.org/10.1109/ICTAI.2004.94

[65] S. M. Majercik. 2007. APPSSAT: Approximate probabilistic planning using stochastic satisfiability. *International Journal of Approximate Reasoning* 45, 2 (2007), 402–419. https://doi.org/10.1016/j.ijar.2006.06.016

[66] S. M. Majercik. 2009. Stochastic Boolean satisfiability. In *Handbook of Satisfiability*. IOS Press, 887–925. https://doi.org/10.3233/978-1-58603-929-5-887

[67] S. M. Majercik and B. Boots. 2005. DC-SSAT: A divide-and-conquer approach to solving stochastic satisfiability problems efficiently. In *Proc. AAAI*. AAAI Press / The MIT Press, 416–422. http://www.aaai.org/Library/AAAI/2005/aaai05-066.php

[68] S. M. Majercik and M. L. Littman. 1998. MAXPLAN: A new approach to

probabilistic planning. In *Proc. AIPS*. AAAI, 86–93. http://www.aaai.org/Library/AIPS/1998/aips98-011.php

[69] S. M. Majercik and M. L. Littman. 2003. Contingent planning under uncertainty via stochastic satisfiability. *Artificial Intelligence* 147, 1-2 (2003), 119–162. https://doi.org/10.1016/S0004-3702(02)00379-X

[70] D. Marculescu, R. Marculescu, and M. Pedram. 1998. Trace-driven steady-state probability estimation in FSMs with application to power estimation. In *Proc. DATE*. IEEE Computer Society, 774–779. https://doi.org/10.1109/DATE.1998.655946

[71] J. P. Marques-Silva and K. A. Sakallah. 2000. Boolean satisfiability in electronic design automation. In *Proc. DAC*. ACM, 675–680. https://doi.org/10.1145/337292.337611

[72] J. Miao, A. Gerstlauer, and M. Orshansky. 2013. Approximate logic synthesis under general error magnitude and frequency constraints. In *Proc. ICCAD*. IEEE, 779–786. https://doi.org/10.1109/ICCAD.2013.6691202

[73] J. Miao, A. Gerstlauer, and M. Orshansky. 2014. Multi-level approximate logic synthesis under general error constraints. In *Proc. ICCAD*. IEEE, 504–510. https://doi.org/10.1109/ICCAD.2014.7001398

[74] A. Mishchenko, S. Chatterjee, R. K. Brayton, and N. Eén. 2006. Improve-

DOI

ments to combinational equivalence checking. In *Proc. ICCAD*. ACM, 836–843. https://doi.org/10.1145/1233501.1233679

[75] N. Miskov-Zivanov and D. Marculescu. 2008. Modeling and optimization for soft-error reliability of sequential circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 5 (2008), 803–816. https://doi.org/10.1109/TCAD.2008.917591

[76] N. Miskov.Zivanov and D. Marculescu. 2006. Circuit reliability analysis using symbolic techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 12 (2006), 2638–2649. https://doi.org/10.1109/TCAD.2006.882592

[77] S. Mitra, M. Zhang, S. Waqas, N. Seifert, B. S. Gill, and K. S. Kim. 2006. Combinational logic soft error correction. In *Proc. ITC*. IEEE Computer Society, 1–9. https://doi.org/10.1109/TEST.2006.297681

[78] K. Mohanram and N. A. Touba. 2003. Cost-effective approach for reducing soft error failure rate in logic circuits. In *Proc. ITC*. IEEE Computer Society, 893–901. https://doi.org/10.1109/TEST.2003.1271075

[79] G. E. Moore. 1965. Cramming more components onto integrated circuits. *Electronics* 38, 8 (1965), 114–117. https://newsroom.intel.com/wp-content/uploads/sites/11/2018/05/moores-law-electronics.pdf

[80] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasícek, and K. Roy. 2016. De-

DOI

sign of power-efficient approximate multipliers for approximate artificial neural networks. In *Proc. ICCAD*. ACM, 81:1–81:7. https://doi.org/10.1145/2966986.2967021

[81] F. N. Najm. 1994. A survey of power estimation techniques in VLSI circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 2, 4 (1994), 446–455. https://doi.org/10.1109/92.335013

[82] M. Narizzano, L. Pulina, and A. Tacchella. 2006. The QBFEVAL web portal. In *Proc. JELIA (LNCS 4160)*. Springer, 494–497. https://doi.org/10.1007/11853886_45

[83] N. J. Nilsson. 2014. *Principles of Artificial Intelligence*. Morgan Kaufmann.

[84] F. A. Oliehoek and C. Amato. 2016. *A Concise Introduction to Decentralized POMDPs*. Springer. https://doi.org/10.1007/978-3-319-28929-8

[85] F. A. Oliehoek, M. T. J. Spaan, and N. A. Vlassis. 2008. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research* 32 (2008), 289–353. https://doi.org/10.1613/jair.2447

[86] C. H. Papadimitriou. 1985. Games against nature. *J. Comput. System Sci.* 31, 2 (1985), 288–301. https://doi.org/10.1016/0022-0000(85)90045-5

[87] G. L. Peterson and J. H. Reif. 1979. Multiple-person alternation. In *Proc. FOCS*. IEEE Computer Society, 348–363. https://doi.org/10.1109/SFCS.1979.25

[88] G. L. Peterson, J. H. Reif, and S. Azhar. 2001. Lower bounds for multiplayer noncooperative games of incomplete information. *Computers & Mathematics with Applications* 41, 7 (2001), 957–992. https://doi.org/10.1016/S0898-1221(00)00333-3

[89] K. Pipatsrisawat and A. Darwiche. 2009. A new d-DNNF-based bound computation algorithm for functional E-MAJSAT. In *Proc. IJCAI*. AAAI Press, 590–595. http://ijcai.org/Proceedings/09/Papers/104.pdf

[90] M. N. Rabe and L. Tentrup. 2015. CAQE: A certifying QBF solver. In *Proc. FMCAD*. IEEE, 136–143. https://doi.org/10.1109/FMCAD.2015.7542263

[91] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel. 2016. Architectural-space exploration of approximate multipliers. In *Proc. ICCAD*. ACM, 80:1–80:8. https://doi.org/10.1145/2966986.2967005

[92] T. Rejimon and S. Bhanja. 2005. Scalable probabilistic computing models using Bayesian networks. In *Proc. MWSCAS*. IEEE, 712–715. https://doi.org/10.1109/MWSCAS.2005.1594200

[93] S. J. Russell and P. Norvig. 2020. *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

[94] R. Salmon and P. Poupart. 2020. On the relationship between stochastic satisfiability and Markov decision processes. In *Proc. UAI*. PMLR, 1105–1115. http://proceedings.mlr.press/v115/salmon20a.html

[95] T. Sang, F. Bacchus, P. Beame, H. A. Kautz, and T. Pitassi. 2004. Combining component caching and clause learning for effective model counting. In *Proc. SAT*. 20–28. http://www.satisfiability.org/SAT04/programme/21.pdf

[96] T. Sang, P. Beame, and H. A. Kautz. 2005. Heuristics for fast exact model counting. In *Proc. SAT (LNCS 3569)*. Springer, 226–240. https://doi.org/10.1007/11499107_17

[97] T. Sang, P. Beame, and H. A. Kautz. 2005. Performing Bayesian inference by weighted model counting. In *Proc. AAAI*. AAAI Press / The MIT Press, 475–482. http://www.aaai.org/Library/AAAI/2005/aaai05-075.php

[98] C. Scholl and R. Wimmer. 2018. Dependency quantified Boolean formulas: An overview of solution methods and applications. In *Proc. SAT (LNCS 10929)*. Springer, 3–16. https://doi.org/10.1007/978-3-319-94144-8_1

[99] R. Schultz. 2003. Stochastic programming with integer variables. *Mathematical Programming* 97, 1-2 (2003), 285–309. https://doi.org/10.1007/s10107-003-0445-z

[100] S. Sinha, A. Mishchenko, and R. K. Brayton. 2002. Topologically constrained logic synthesis. In *Proc. ICCAD*. ACM / IEEE Computer Society, 679–686. https://doi.org/10.1145/774572.774672

[101] F. Somenzi. [n.d.]. CUDD: CU decision diagram package. http://vlsi.colorado.edu/ fabio/.

[102] L. J. Stockmeyer and A. R. Meyer. 1973. Word problems requiring exponential time: Preliminary report. In *Proc. STOC*. ACM, 1–9. https://doi.org/10.1145/800125.804029

[103] T. Teige and M. Fränzle. 2010. Resolution for stochastic Boolean satisfiability. In *Proc. LPAR (LNCS 6397)*. Springer, 625–639. https://doi.org/10.1007/978-3-642-16242-8_44

[104] L. Tentrup and M. N. Rabe. 2019. Clausal abstraction for DQBF. In *Proc. SAT (LNCS 11628)*. Springer, 388–405. https://doi.org/10.1007/978-3-030-24258-9_27

[105] G. S. Tseitin. 1983. On the complexity of derivation in propositional calculus. In *Automation of Reasoning*. Springer, 466–483. https://doi.org/10.1007/978-3-642-81955-1_28

[106] S. Venkataramani, A. Sabne, V. J. Kozhikkottu, K. Roy, and A. Raghunathan. 2012. SALSA: Systematic logic synthesis of approximate circuits. In *Proc. DAC*. ACM, 796–801. https://doi.org/10.1145/2228360.2228504

[107] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. 2011. MACACO: Modeling and analysis of circuits for approximate computing. In *Proc. ICCAD*. IEEE Computer Society, 667–673. https://doi.org/10.1109/ICCAD.2011.6105401

[108] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng. 2009. *Electronic Design Automation: Synthesis, Verification, and Test* (1st ed.). Morgan Kaufmann.

[109] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu. 2013. On reconfiguration-oriented approximate adder design and its application. In *Proc. ICCAD*. IEEE, 48–54. https://doi.org/10.1109/ICCAD.2013.6691096