A Sharp Leap from Quantified Boolean Formula to Stochastic Boolean Satisfiability Solving

Pei-Wei Chen,[1] **Yu-Ching Huang,**[1] **Jie-Hong R. Jiang**[1,2]
[1] Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan
[2] Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 10617, Taiwan
perrypeiweichen@gmail.com, nk930439gj94@gmail.com, jhjiang@ntu.edu.tw

Abstract

Stochastic Boolean Satisfiability (SSAT) is a powerful representation for the concise encoding of quantified decision problems with uncertainty. While it shares commonalities with quantified Boolean formula (QBF) satisfiability and has the same PSPACE-complete complexity, SSAT solving tends to be more challenging as it involves expensive model counting, a.k.a. Sharp-SAT. To date, SSAT solvers, especially those imposing no restrictions on quantification levels, remain much lacking. In this paper, we present a new SSAT solver based on the framework of clause selection and cube distribution previously proposed for QBF solving. With model counting integrated and learning techniques strengthened, our solver is general and effective. Experimental results demonstrate the overall superiority of the proposed algorithm in both solving performance and memory usage compared to the state-of-the-art solvers on a number of benchmark formulas.

1 Introduction

Stochastic Boolean Satisfiability (SSAT) is a formulation of *games against nature* (Papadimitriou 1985; Majercik 2009). While it is a generalization of the satisfiability of Quantified Boolean Formula (QBF) with the addition of randomized quantification, its computational complexity remains the same as QBF in the PSPACE-complete class. On the one hand, the generality of SSAT makes it able to concisely encode many interesting decision problems with uncertainty, such as probabilistic planning (Littman, Majercik, and Pitassi 2001), trust management (Freudenthal and Karamcheti 2003), belief network inference (Littman, Majercik, and Pitassi 2001), and probabilistic equivalence checking (Lee and Jiang 2018). On the other hand, the randomized quantification of SSAT imposes computation sophistication due to its counting nature in calculating the satisfying probability of a formula. It involves not just Boolean Satisfiability (SAT), asking *are there solutions*, but the Sharp Satisfiability (Sharp-SAT), a.k.a. Model Counting, asking *how many solutions*. As Sharp-SAT is #P-complete, its computation is considered more challenging than the SAT problem of NP-completeness. This challenge is due to the powerfulness of a counting oracle as manifested by Toda's Theorem (Toda 1991), which states that any problem in the Polynomial Hierarchy (PH) can be reduced in polynomial time to a counting problem for a single query to a #P-oracle, and thus $\mathrm{PH} \subseteq \mathrm{P}^{\#\mathrm{P}}$.

To date, there are only a few SSAT solvers publicly available. Among the state-of-the-art solvers, `DC-SSAT` (Majercik and Boots 2005), a Davis-Putnam-Logemann-Loveland (DPLL)-based SSAT solver, divides the problem into subproblems and conquers them by exploiting structural characteristics of completely observable probabilistic planning (COPP) problems. `erSSAT` (Lee, Wang, and Jiang 2018) takes advantage of the idea of *clause selection* (Janota and Marques-Silva 2015), a QBF solving technique, to effectively prune the search space of exist-random quantified SSAT formulas, which is known as E-MAJSAT (Littman, Goldsmith, and Mundhenk 1998). `reSSAT` (Lee, Wang, and Jiang 2017) utilizes modern SAT solvers and model counters, and employs the generalization of assignments to efficiently explore the search space of random-exist quantified SSAT formulas. Among the above, `DC-SSAT` is the only solver that can cope with general SSAT formulas, without restricting the quantification structure. In this work, we are concerned with solving general SSAT formulas.

Unlike `DC-SSAT`, a DPLL-based search algorithm implemented from scratch, our algorithm takes an off-the-shelf SAT solver and model counter as blackbox subroutines for computation. Inspired by `erSSAT` and `reSSAT`, which incorporate clause selection and model counting into SSAT solving for two quantification levels, we devise a general algorithm and overcome the level limitation. Devising the pruning technique and counting method for randomized quantification levels is the key to general SSAT solving. Different from `erSSAT`, which only applies pruning techniques at existential levels, our approach additionally applies pruning at randomized levels. The pruning technique and counting method at randomized levels are also different from those of `reSSAT`. Particularly `reSSAT` uses minterm generalization and the return value of a subproblem is restricted to either 0 or 1. In contrast, we calculate the probability based on clause selection and deal with return values of arbitrary probability constant between 0 and 1. These differences overcome the limitations of `erSSAT` and `reSSAT` on two-level SSAT formulas and allow us to solve general multi-level SSAT formulas. Combined with several enhancement techniques, the performance of the proposed algorithm is further improved. As the algorithm uses a SAT solver and model counter as standalone engines, it can directly profit from the advancement of these solvers without modifications. Moreover, the algorithm can be easily modified for *incomplete SSAT* by deriving lower and upper bounds to the exact solution. With evaluation on a variety of benchmarks, experimental results show the overall superiority of the proposed algorithm in both solving performance and memory usage compared to the state-of-the-art solvers.

2 Preliminaries

For Boolean connectives, we denote conjunction by "$\wedge$" (sometimes "$\cdot$" or even being omitted in an expression for brevity), disjunction by "$\vee$", biconditional by "$\leftrightarrow$" (or "$\equiv$"), and negation by "$\neg$" (or an overline). For Boolean values, TRUE and FALSE are denoted by "$\top$" and "$\bot$", respectively. In a Boolean formula, a *literal* is either a variable (referred to as a *positive-phase* literal) or the negation of a variable (referred to as a *negative-phase* literal); a *cube* is a conjunction of literals; a *clause* is a disjunction of literals. A Boolean formula is in the *conjunction normal form* (CNF) if it is expressed as a conjunction of clauses. A cube is referred to as a *minterm* with respect to a set of variables if all variables in the set appear in the cube. The corresponding variable of a literal l is denoted as $\mathsf{var}(l)$. For notational convenience, we treat a cube and a clause as a set of literals, and a CNF formula as a set of clauses.

A Boolean formula ϕ over a set of variables X defines a unique Boolean function $\mathbb{B}^{|X|} \to \mathbb{B}$, where $|X|$ is the cardinality of X. An *assignment* τ over a set of variables X, denoted as $\tau(X)$, is a mapping $\tau : X \to \mathbb{B}$. An assignment τ is called *full* if every variable $x \in X$ is mapped by τ to some Boolean value, i.e., $\tau(x) \in \{\bot, \top\}$; otherwise, it is called *partial*. We alternatively treat an assignment $\tau(X)$ as a cube consisting of literals $\{l \mid l = x \text{ for } \tau(x) = \top, l = \neg x \text{ for } \tau(x) = \bot, \text{ and } x \in X\}$. By abusing the notation, we use $x \in \tau$ to mean $\tau(x) = \top$ and $\neg x \in \tau$ to mean $\tau(x) = \bot$. The application of an assignment τ to a Boolean formula ϕ, called *cofactoring*, results in a new formula ϕ' obtained by substituting every occurrence of each variable x in ϕ with Boolean value $\tau(x)$. Given a Boolean formula ϕ and a cube c, the *cofactor* of ϕ on c, denoted as $\phi|_c$, is derived by iteratively cofactoring ϕ on each literal $l \in c$.

2.1 Stochastic Boolean Satisfiability

An SSAT formula over variables $X = \bigcup_{i=1}^{i=n} X_i$, with $X_i \neq \emptyset$, $X_i \cap X_j = \emptyset$ for $i \neq j$, can be expressed in the prenex form

$$\Phi = Q_1 X_1, ..., Q_n X_n . \phi(X_1, ..., X_n) \tag{1}$$

where $Q_1 X_1 ... Q_n X_n$, for $Q_i \in \{\exists, \text{Ⅎ}\}$ being either an existential $\exists$ or randomized Ⅎ quantifier and $Q_i \neq Q_{i+1}$, is called the *prefix*, and ϕ, a quantifier-free Boolean formula, is called the *matrix*. We denote the sets of existentially and randomly quantified variables as $X_\exists$ and $X_{\text{Ⅎ}}$, respectively. For variable $x \in X_i$, we define the *quantification level* of x, denoted $\mathsf{level}(x)$, to be i. We also extend the notion of quantification level to a literal l, with $\mathsf{level}(l)$ meaning $\mathsf{level}(\mathsf{var}(l))$. When a randomized quantifier is applied on a variable $x \in X_{\text{Ⅎ}}$, it is associated with a probability p_x in interval $[0, 1]$, denoted as $\text{Ⅎ}^{p_x} x$, indicating $x = \top$ and $\bot$ with probabilities p_x and $(1 - p_x)$, respectively. In the sequel, we shall assume ϕ being expressed in CNF.

The semantics of an SSAT formula Φ is concerned with its expectation of satisfaction by the following interpretation. Let x be the outermost variable in the prefix of Φ. Then the satisfying probability of Φ, denoted as $\mathsf{Pr}[\Phi]$, can be computed recursively by the rules:

1. $\mathsf{Pr}[\top] = 1$,
2. $\mathsf{Pr}[\bot] = 0$,
3. $\mathsf{Pr}[\Phi] = \max\{\mathsf{Pr}[\Phi|_{\neg x}], \mathsf{Pr}[\Phi|_x]\}$, if $x \in X_\exists$
4. $\mathsf{Pr}[\Phi] = (1 - p_x)\,\mathsf{Pr}[\Phi|_{\neg x}] + p_x\,\mathsf{Pr}[\Phi|_x]$, if $x \in X_{\text{Ⅎ}}$

We remark that an SSAT formula can be extended by further allowing the universal quantifier $\forall$. This extension, however, does not affect the PSPACE-complete computation complexity of SSAT. Under this extension, the quantified Boolean formula (QBF) is a special case of SSAT without randomized quantifiers. As the extension does not change much the formulation, for simplicity we focus on solving SSAT formulas involving only existential and randomized quantifiers.

2.2 Model Counting

Given a CNF formula ϕ over variables X, the *model counting*, or *Sharp-SAT*, problem asks how many satisfying assignments are there. Generally, one may ask a *weighted* version of model counting with respect to some weight function $\omega : L_X \to N$, where $L_X = \{x, \neg x \mid x \in X\}$ is the literal set of X and N is a set of weight values. The weight of an assignment τ is defined as the product of the weights of the literals in τ. In this work, the weight function is specialized with $\omega(x) \in [0, 1]$, and $\omega(\neg x) = 1 - \omega(x)$ for any $x \in X$. Thereby, $\omega(x)$ corresponds to the probability $\Pr[x = \top]$, and the summation of the weights of all satisfying assignments of a given CNF formula corresponds to its satisfying probability with respect to ω.

A model counting algorithm can be *exact* (Sang et al. 2004; Sang, Beame, and Kautz 2005) or *approximate* (Gomes, Sabharwal, and Selman 2006; Gomes et al. 2007; Chakraborty, Meel, and Vardi 2016) depending on whether it gives an exact answer. An approximate model counter may possibly provide some upper and/or lower bound on the weight summation of satisfying assignments with some confidence level.

2.3 Clause Selection in QBF Solving

Clause selection (Janota and Marques-Silva 2015) is a QBF solving technique to track the clause satisfaction status and to facilitate learning using abstract variables. Given a QBF with its matrix $\phi = C_1 \wedge \cdots \wedge C_n$, the subclause of a clause C_i consisting of literals $\{l \in C_i \mid \mathsf{level}(l) \bowtie k\}$ with respect to some k is denoted as $C_i^{\bowtie k}$, where $\bowtie \in \{=, <, \leq, >, \geq\}$. For conciseness, we abbreviate $C_i^{=k}$ as C_i^k in the sequel.

A clause C_i is said to be *selected* at quantification level k if all literals in $C_i^{\leq k}$ are valuated to $\bot$; otherwise, C_i is said to be *deselected* at quantification level k. Note that once a clause is deselected at quantification level j, it remains deselected at quantification levels greater than j, regardless of the valuations of the literals in $C_i^{>j}$. To track whether clauses have been deselected, for each quantification level j a *selection variable* $s_i^j \equiv \neg C_i^{\leq j}$ is introduced for each clause C_i.

The way clause selection works can be intuitively explained under the game interpretation of QBF played between the $\exists$-player and the $\forall$-player. In round j with $Q_j = \exists$ (resp. $\forall$) for $j = 1, \ldots, n$ in order, the $\exists$-player (resp. $\forall$-player) assigns the variables in X_j, with the intention to satisfy (resp. falsify) the matrix. The QBF is true if and only if there exists a winning strategy for the $\exists$-player that satisfies the matrix, regardless of how the $\forall$-player plays. The clause selection technique explores the search space by finding possible selection statuses of clauses and adding learnt information to exclude failing strategies for each player. For example, if the matrix evaluates to $\perp$, meaning that there is a set S_C of clauses which remain selected at some round of the game, the $\exists$-player loses under the current assignments. To rectify the strategy of the $\exists$-player, the clause selection technique backtracks to some previous level and adds a learnt constraint to enforce a deselection of at least one clause in S_C at the level.

3 From QBF to SSAT Solving

A key observation that enables the extension of the clause selection framework from QBF to SSAT is explained as follows. In contrast to QBF solving, for clause-selection-based SSAT solving, one must explore all possible assignments at a randomly quantified level before returning to some previous level. Also, the learning and backtracking conditions in SSAT are much more stringent than those in QBF. To make SSAT solving feasible, we introduce the *local selection variable* $t_i^j \equiv \neg C_i^j$, which checks whether the valuations of literals in C_i^j deselects C_i. If all literals in C_i^j are valuated to $\perp$, we say that C_i is *locally selected* at level j; otherwise, it is *locally deselected* at level j.

Let T_j be the set of local selection variables at level j. The formula $\psi_j(X_j, T_j) = \bigwedge_{C_i \in \phi}(t_i^j \equiv \neg C_i^j)$ is called the *selection relation* of ϕ. The application of an assignment $\tau_j(X_j)$ to ϕ corresponds to a selection status of clauses at quantification level j, which can be described by a *selection minterm* $m_{T_j} = \psi_j|_{\tau_j}$, obtained by applying τ_j to ψ_j. A *selection cube* c_{T_j} is a selection minterm with some literals being removed subject to retaining the same satisfying probability (to be formally stated in Property 2).

Example 1. *Consider the SSAT formula* Φ*:*

$$\text{Я}^{0.5}x_1, \text{Я}^{0.7}x_2, \exists y_1, \exists y_2, \exists y_3, \text{Я}^{0.4}z_1, \text{Я}^{0.8}z_2.$$
$$(\overline{x_1} \vee \overline{y_2} \vee z_1)(\overline{y_3} \vee \overline{z_1})(x_1 \vee y_1 \vee y_2 \vee y_3)(\overline{x_2} \vee y_2 \vee \overline{z_2})$$

over variables $X_1 = \{x_1, x_2\}$*,* $X_2 = \{y_1, y_2, y_3\}$*, and* $X_3 = \{z_1, z_2\}$ *in three quantification levels, and with four clauses in the matrix. For each quantification level j, a local selection variable* t_i^j *is introduced for each clause* C_i*. The selection relation at level 1, for example, is*

$$\psi_1 = (t_1^1 \equiv x_1)(t_2^1 \equiv \top)(t_3^1 \equiv \neg x_1)(t_4^1 \equiv x_2).$$

The assignment $\tau_1 = x_1 \neg x_2$ *over* X_1 *locally selects* C_1 *and* C_2 *and deselects* C_3 *and* C_4*. The selection status can be seen from* $m_{T_1} = \psi_1|_{\tau_1} = t_1^1 t_2^1 \neg t_3^1 \neg t_4^1$*.*

4 Algorithm Overview

Consider an SSAT formula of Eq. (1). For each quantification level j, we maintain a SAT routine to work on solving the selection relation ψ_j. The solving process is performed recursively. Starting from level 1, we obtain an assignment τ_1 over X_1 from solving ψ_1 and apply it to Φ, which produces a subproblem $\Phi' = Q_2X_2, \ldots, Q_nX_n.\phi|_{\tau_1}$. By recursively solving Φ', it returns a probability to the first level. We then add a learnt clause to ψ_1 and create another subproblem Φ'' if $\mathsf{SAT}(\psi_1) = \top$; otherwise, the space spanned by variables X_1 is completely searched, and the resulting satisfying probability p is returned. The same procedure is performed for each subproblem. Depending on the quantification type of Q_1, different operations are done to obtain the learnt clause and the returned probability, as detailed in Algorithms 1 and 2, to be elaborated in Sections 5 and 6, respectively.

Extended from a similar statement in the context of E-MAJSAT in (Lee, Wang, and Jiang 2018), Property 1 holds for general SSAT formulas, and allows effectively search space pruning for both existential and randomized levels.

Property 1 (Matrix Containment Property)**.** *For two SSAT formulas* $\Phi_1 = Q_1X_1, ..., Q_nX_n.\phi_1$ *and* $\Phi_2 = Q_1X_1, ..., Q_nX_n.\phi_2$ *sharing the same prefix, if* $\phi_1 \subseteq \phi_2$*, then* $\mathsf{Pr}[\Phi_2] \leq \mathsf{Pr}[\Phi_1]$.

For $\phi_1 \subseteq \phi_2$, any assignment satisfying ϕ_2 satisfies ϕ_1. Hence all assignments contributing to $\mathsf{Pr}[\Phi_2]$ also contributes to $\mathsf{Pr}[\Phi_1]$. Observe that Property 1 holds regardless of the quantifier types at each level. It plays a key role in the following sections.

Note that in the following sections, a considered SSAT formula can be an induced formula after some variables being assigned. That is, only the selected clauses and unassigned variables remain in the considered formula.

5 Solving Existentially Quantified Levels

Consider an SSAT formula of the form

$$\Phi = \exists X_1, ..., Q_nX_n.\phi$$

To compute the satisfying probability of Φ, it suffices to enumerate and apply all possible assignments $\tau(X_1)$ and solve the induced subproblems $\Phi|_\tau$. Clearly, this brute-force approach is computationally expensive. Extending the idea from the E-MAJSAT solver `erSSAT` to cope with multi-level SSAT formulas, this problem can be solved more efficiently with clause selection introduced.

Consider an assignment $\tau_1(X_1)$ and its application to ϕ which is $\phi|_{\tau_1}$. For any other assignments $\tau_2(X_1)$ where $\phi|_{\tau_1} \subseteq \phi|_{\tau_2}$, by Property 1, we get $\mathsf{Pr}[\Phi|_{\tau_2}] \leq \mathsf{Pr}[\Phi|_{\tau_1}]$. Because $Q_1 = \exists$ and $\mathsf{Pr}[\Phi|_{\tau_2}] \leq \mathsf{Pr}[\Phi|_{\tau_1}]$, assignment τ_2

Algorithm 1 *SolveSSAT*-$\exists(\Phi)$

Input: $\Phi : \Phi = \exists X_1 ... Q_n X_n.\phi$ where $Q_i \in \{\exists, \text{ꓤ}\}$
Output: $p_{\max}$: the satisfying probability of Φ,
$\tau_{\max}$: the assignment over X_1 s.t. $\mathsf{Pr}[\Phi|_{\tau_{\max}}] = p_{\max}$
1: $p_{\max} := 0$
2: $\tau_{\max} := \emptyset$
3: **if** $n = 1$ // Last level
4:     **if** $\mathsf{SAT}(\phi) = \top$
5:         $p_{\max} := 1$
6:         $\tau_{\max} :=$ the found model of ϕ
7: **else**
8:     $\psi_1(X_1, T_1) := \bigwedge_{C_i \in \phi}(t_i^1 \equiv \neg C_i^1)$
9:     **while** $\mathsf{SAT}(\psi_1) = \top$
10:         $\tau :=$ the found model of ψ_1 for variables in X_1
11:         $p :=$ *SolveSSAT*-$\text{ꓤ}(\Phi|_\tau)$
12:         **if** $p > p_{\max}$
13:             $p_{\max} := p$
14:             $\tau_{\max} := \tau$
15:         $c_{T_1} :=$ *RemoveNegativeLits*$(\psi_1|_\tau)$
16:         $C_L := \neg c_{T_1}$
17:         $\psi_1 := \psi_1 \wedge C_L$
18:         **if** $p = 0$
19:             *AddLearntClausesToPriorLevels*(C_L)
20: **return** $(p_{\max}, \tau_{\max})$

needs not be explored if τ_1 has been explored. For all such assignments τ_2, they should be blocked once τ_1 is explored.

To prevent from obtaining assignment τ_2 such that $\phi|_{\tau_2}$ is a superset of $\phi|_{\tau_1}$, at least one of the clauses in $\phi|_{\tau_1}$ should be deselected. A learnt clause C_L, which can be obtained by negating the selection minterm $m_{T_1} = \psi_1|_{\tau_1}$ and keeping the negative-phase literals, is added to ψ_1 to enforce the selection. The largest satisfying probability of subproblems and the corresponding assignments to existential variables are kept throughout the process and returned when $\mathsf{SAT}(\psi_1) = \bot$. Algorithm 1 sketches the procedure in detail, where the subroutine *RemoveNegativeLits* in line 15 obtains the selection cube by removing the negative-phase literals in the selection minterm $\psi_1|_\tau$, and line 19 runs an enhancement technique to be discussed in Section 7.

Example 2. *Continue Example 1. The subproblem $\Phi|_\tau$, for $\tau = x_1 x_2$, equals*

$$\exists y_1, \exists y_2, \exists y_3, \text{ꓤ}^{0.4} z_1, \text{ꓤ}^{0.8} z_2.(\overline{y_2} \vee z_1)(\overline{y_3} \vee \overline{z_1})(y_2 \vee \overline{z_2}),$$

and the selection relation ψ_2 at the second level is

$$\psi_2 = (t_1^2 \equiv y_2)(t_2^2 \equiv y_3)(t_3^2 \equiv \neg y_1 \neg y_2 \neg y_3)(t_4^2 \equiv \neg y_2).$$

Suppose the first tried partial assignment $\tau_1(X_2)$ is $\tau_1 = \neg y_1 \neg y_2 \neg y_3$, which locally deselects C_1 and C_2 and $m_{T_2} = \psi_2|_{\tau_1} = \neg t_1^2 \neg t_2^2 t_3^2 t_4^2$. By invoking the weighted model counter on the subproblem $\Phi|_{\tau\tau_1} = \text{ꓤ}^{0.8} z_2.(\neg z_2)$, we obtain $\mathsf{Pr}[\Phi|_{\tau\tau_1}] = 0.2$. The learnt clause $C_L = (\neg t_3^2 \vee \neg t_4^2)$ is then added to ψ_2 to prevent C_1 and C_2 from being deselected simultaneously. Note that $\neg t_3^2$ can be discarded from C_L since C_3 is already deselected at the first level. Suppose the next tried partial assignment is $\tau_2 = \neg y_1 y_2 \neg y_3$, which locally deselects C_2, C_3, and C_4. The satisfying probability of the subproblem $\Phi|_{\tau\tau_2} = \text{ꓤ}^{0.4} z_1.(z_1)$ equals 0.4, and another learnt clause $C_L = (\neg t_1^2)$ is added to ψ_2. As $\mathsf{SAT}(\psi_2) = \bot$, the process ends and we conclude that $\mathsf{Pr}[\Phi|_\tau] = 0.4$.

Algorithm 2 *SolveSSAT*-$\text{ꓤ}(\Phi)$

Input: $\Phi : \Phi = \text{ꓤ} X_1 ... Q_n X_n.\phi$ where $Q_i \in \{\exists, \text{ꓤ}\}$
Output: p : the satisfying probability of Φ
1: $p := 0$
2: **if** $n = 1$ // Last level
3:     **if** $\mathsf{SAT}(\phi) = \top$
4:         $p :=$ *WeightedModelCount*$(\text{ꓤ} X_1.\phi)$
5: **else**
6:     $V := \emptyset$
7:     $\psi_1(X_1, T_1) := \bigwedge_{C_i \in \phi}(t_i^1 \equiv \neg C_i^1)$
8:     $\psi_2(X_2, T_2) := \bigwedge_{C_i \in \phi}(t_i^2 \equiv \neg C_i^2)$
9:     **while** $\mathsf{SAT}(\psi_1) = \top$
10:         $\tau_1 :=$ the found model of ψ_1 for variables in X_1
11:         $(p, \tau_2) :=$ *SolveSSAT*-$\exists(\Phi|_\tau)$
12:         $\tau_2' :=$ *MaximalPruning*$(\psi_1|_{\tau_1}, \psi_2, \tau_2)$
13:         $c_{T_1} :=$ *PruneSelection*$(\psi_1|_{\tau_1}, \psi_2|_{\tau_2'})$
14:         *V.CollectProbabilitySelectionCubesPair*(p, c_{T_1})
15:         $C_L := \neg c_{T_1}$
16:         $\psi_1 := \psi_1 \wedge C_L$
17:         **if** $p = 0$
18:             *AddLearntClausesToPriorLevels*(C_L)
19:     $p :=$ *ComputeProbability*(V)
20: **return** p

6 Solving Randomly Quantified Levels

Consider an SSAT formula of the form

$$\Phi = \text{ꓤ} X_1, ..., Q_n X_n.\phi$$

Since randomly quantified levels require to compute the weighted sum of $\mathsf{Pr}[\Phi|_\tau]$ with weight $\omega(\tau)$ over all possible assignments $\tau(X_1)$, the difficulty in solving such formula lies in the exponentially growing number of possible assignments. Based on the clause selection framework, we propose our solution below.

Notice that the valuations of different assignments may result in the same selection of clauses, thus the same subproblem. According to this observation, instead of enumerating all possible assignments, we could list the possible selections of clauses, represented as selection cubes, and solve the corresponding subproblems. Similar to Section 5, this can be done by solving and adding learnt clauses, which block the previously searched selection cubes, to ψ_1 until $\mathsf{SAT}(\psi_1) = \bot$. However, since randomly quantified levels require to compute the aggregated satisfying probability of all assignments, the pruning technique in Section 5 cannot be applied. At the end of the solving process, all searched selection cubes and the return values of the corresponding subproblems are used to compute the satisfying probability of Φ, which will be explained in Section 6.2. The solving procedure is made precise in Algorithm 2. The subroutine *PruneSelection* in line 13 is the pruning technique to be explained in Section 6.1, and lines 14 and 19 perform intermediate information collection and satisfying probability computation as to be detailed in Section 6.2. Also, lines 12 and 18 are enhancement techniques to be presented in Section 7.

Notice that the number of selection cubes could be of exponential size in the number of clauses. To accelerate the solving process, we propose the pruning technique described in Section 6.1 to effectively prune the search space.

6.1 Pruning in Randomly Quantified Levels

In this section, we take two quantification levels, $\reflectbox{R}X_1\exists X_2$, into account and take advantage of the following property to prune the search space.

Property 2 (Selection Pruning Property)**.** *Given an SSAT formula*

$$\Phi = \reflectbox{R}X_1\exists X_2, ..., Q_nX_n.\phi \tag{2}$$

and selection minterms $m_{T_1} = \psi_1|_{\tau_1}$ and $m_{T_2} = \psi_2|_{\tau_2}$, with τ_1 over X_1 and τ_2 over X_2, assume τ_2 gives the maximum probability $p_{\max}$ of subproblem $\Phi|_{\tau_1}$, i.e., $\mathsf{Pr}[\Phi|_{\tau_1}] = \mathsf{Pr}[\Phi|_{\tau_1\tau_2}] = p_{\max}$. If $\neg t_i^1 \in m_{T_1}$ and $\neg t_i^2 \in m_{T_2}$, then $\mathsf{Pr}[\Phi|_{\tau_1'}] = p_{\max}$, where $\psi_1|_{\tau_1'} = m_{T_1}' = c_{T_1} \cup \{t_i^1\}$, for $c_{T_1} = m_{T_1} \setminus \{\neg t_i^1\}$.

The correctness of the property can be understood by the following observation. First, since $\phi|_{\tau_1} \subseteq \phi|_{\tau_1'}$, by Property 1 we know

$$\mathsf{Pr}[\Phi|_{\tau_1'}] \le \mathsf{Pr}[\Phi|_{\tau_1}] = p_{\max} \tag{3}$$

which serves as the upper bound. To check whether $\mathsf{Pr}[\Phi|_{\tau_1'}]$ attends its upper bound, we look at the subproblem $\Phi|_{\tau_1'}$. Notice that since $\Phi|_{\tau_1}$ and $\Phi|_{\tau_1'}$ differ by a clause C_i and τ_2 locally deselects C_i, applying τ_2 to $\Phi|_{\tau_1'}$ results in the same subproblem as $\Phi|_{\tau_1\tau_2}$, i.e. $\Phi|_{\tau_1'\tau_2} = \Phi|_{\tau_1\tau_2}$. Considering that $\mathsf{Pr}[\Phi|_{\tau_1'\tau_2}] = \mathsf{Pr}[\Phi|_{\tau_1\tau_2}] = p_{\max}$ and $Q_2 = \exists$, we obtain

$$\mathsf{Pr}[\Phi|_{\tau_1'}] \ge p_{\max} \tag{4}$$

Hence, from Eq. (3) and (4), we get $\mathsf{Pr}[\Phi|_{\tau_1'}] = p_{\max}$.

Property 2 can be exploited to prune literals from a selection minterm to form a selection cube (as mentioned in Section 3) as follows. After obtaining the assignment τ_2 which maximizes the satisfying probability of subproblem $\Phi|_{\tau_1}$, if a clause C_i is locally deselected by both τ_1 and τ_2 at levels 1 and 2, respectively, for $Q_1 = \reflectbox{R}$ and $Q_2 = \exists$, we can deduce that selecting C_i at level 1 while keeping the selection status of other clauses unchanged leads to the same satisfying probability. That is, whether or not we select C_i at level 1 does not affect the satisfying probability. According to this observation, we can remove the local selection literals that satisfy the above conditions from m_{T_1} to obtain a selection cube c_{T_1} with fewer literals.

Also, consider the case where $p_{\max} = 0$ or 1. If $p_{\max} = 0$ (resp. 1), selecting (resp. deselecting) the originally deselected (resp. selected) clauses at the first level will result in satisfying probability $p \le 0$ (resp. $p \ge 1$). Hence, the negative-phase (resp. positive-phase) literals in m_{T_1} can be removed.

The above operations are done by the *PruneSelection* subroutine in line 13 of Algorithm 2. The resulting selection cube c_{T_1} in line 13 associated with the satisfying probability p in line 11 is added to a set S_p. The pair (p, S_p) is then collected as a vector V, as in line 14 of Algorithm 2. A stronger learnt clause C_L is then obtained by negating c_{T_1} and added to ψ_1 in lines 15 and 16, respectively, of Algorithm 2.

6.2 Weight Computation

In Algorithm 2 line 9, upon $\mathsf{SAT}(\psi_1) = \bot$, the solving process ends and the pairs (p, S_p) stored in V are used to compute $\mathsf{Pr}[\Phi]$ as described in Theorem 1.

Theorem 1. *Given a vector V of pairs (p, S_p) that are collected until $\mathsf{SAT}(\psi_1) = \bot$ for selection relation ψ_1 at level 1, the satisfying probability of Φ can be computed by*

$$\mathsf{Pr}[\Phi] = \sum_{(p,S_p)\in V} p \times \mathsf{Pr}[\reflectbox{R}X_1.\varphi] \tag{5}$$

with

$$\varphi = \bigvee_{c_{T_1} \in S_p} (\exists T_1.\psi_1|_{c_{T_1}}) \tag{6}$$

where X_1 and T_1 are the set of variables and the set of local selection variables at quantification level 1.

Proof. First, since $\mathsf{SAT}(\psi_1) = \bot$, all possible selections of clauses, or selection minterms, have been explored. To compute $\mathsf{Pr}[\Phi]$, we take the weighted summation of all probabilities p collected in V. Each probability p is weighted by w_p, which is the probability of assignments $\tau(X_1)$ to produce subproblems Φ' for $\mathsf{Pr}[\Phi'] = p$. Each of such assignments $\tau(X_1)$ corresponds to a selection minterm m_{T_1}. Observe that $\tau(X_1)$ satisfies $\theta = \exists T_1.\psi_1|_{c_{T_1}}$, where c_{T_1} is a selection cube obtained from $\psi_1|_\tau = m_{T_1}$. That is, all assignments $\tau(X_1)$ that produce some m_{T_1} covered by c_{T_1} satisfy θ. Since all selection minterms associated with satisfying probability p are covered by some selection cubes $c_{T_1} \in S_p$, the disjunction of θ with respect to each $c_{T_1} \in S_p$, expressed as φ in Eq. (6), exactly characterizes all assignments that produce such selection minterms. Note that despite a selection minterm may be covered by multiple selection cubes, the assignments producing that selection minterm are only counted once. Thus, by randomly quantifying X_1 and invoking a model counter, we get $w_p = \mathsf{Pr}[\reflectbox{R}X_1.\varphi]$. Finally, by taking the weighted summation as in Eq. (5), we get $\mathsf{Pr}[\Phi]$. The computation of the satisfying probability is done by *ComputeProbability* in line 19 of Algorithm 2. ☐

Example 3. *Consider the SSAT formula Φ and the selection relation ψ_1 in Example 1. Let the first tried assignment τ_1 be $\neg x_1\neg x_2$, which selects C_2 and C_3 and deselects C_1 and C_4. As assignment $\tau = \neg y_1 y_2 \neg y_3$ satisfies the subproblem $\Phi|_{\tau_1}$, we get $\mathsf{Pr}[\Phi|_{\tau_1}] = 1$. Now, consider two selection minterms $m_{T_1} = \psi_1|_{\tau_1} = \neg t_1^1 t_2^1 t_3^1 \neg t_4^1$ and $m_{T_2} = \psi_2|_\tau = t_1^2\neg t_2^2\neg t_3^2\neg t_4^2$. Since $\neg t_4^1 \in m_{T_1}$ and $\neg t_4^2 \in m_{T_2}$, $\neg t_4^1$ can be removed from m_{T_1}. Moreover, since $\mathsf{Pr}[\Phi|_{\tau_1}] = 1$, the learnt clause is strengthened as $C_L = (t_1^1)$. Let the second tried assignment τ_2 be $x_1\neg x_2$, which selects C_1 and C_2 and deselects C_3 and C_4. As assignment $\tau = y_1\neg y_2\neg y_3$ satisfies the subproblem $\Phi|_{\tau_2}$, we get $\mathsf{Pr}[\Phi|_{\tau_2}] = 1$. Similar to the process above, since $\neg t_3^1 \in m_{T_1} = \psi_1|_{\tau_2} = t_1^1 t_2^1 \neg t_3^1 \neg t_4^1$ and $\neg t_3^2 \in m_{T_2} = \psi_2|_\tau = \neg t_1^2 \neg t_2^2 \neg t_3^2 t_4^2$ and $\mathsf{Pr}[\Phi|_{\tau_2}] = 1$, the learnt clause could be strengthened as $C_L = (t_4^1)$. Let the third tried assignment τ_3 be x_1x_2, which by applying it to Φ produces the subproblem described in Example 2. From Example 2, we know that $\tau = \neg y_1 y_2 \neg y_3$ gives the maximum satisfying probability $\mathsf{Pr}[\Phi|_{\tau_3}] = 0.4$. Since $\neg t_3^1 \in m_{T_1} =$*

$\psi_1|_{\tau_2} = t_1^1 t_2^1 \neg t_3^1 t_4^1$ *and* $\neg t_3^2 \in m_{T_2} = \psi_2|_\tau = t_1^2 \neg t_2^2 \neg t_3^2 \neg t_4^2$, *we obtain the learnt clause* $C_L = (\neg t_1^1 \vee \neg t_2^1 \vee \neg t_4^1)$. *As* $\mathsf{SAT}(\psi_1) = \bot$, *the resulting vector* V *we get is* $V = \{(1, \{\neg t_1^1, \neg t_4^1\}), (0.4, \{t_1^1 t_2^1 t_4^1\})\}$ *where the selection cubes are obtained by negating the learnt clauses. Finally, we obtain* $\mathsf{Pr}[\Phi] = 0.79$ *by Eq.* (5).

7 Enhancement Techniques

The performance of clause-selection-based approach is deeply affected by the strength of the learnt clauses. We introduce three enhancement techniques, 1) *cube distribution*, 2) *maximal clause pruning*, and 3) *non-chronological backtracking*, to further enhance the learning ability.

Cube Distribution: In (Chen and Jiang 2019), a cube-distribution-based QBF solver CUED is proposed, which interprets QBF solving as a process of distributing cubes (clause selection conditions) into the onsets and offsets of Skolem functions. It effectively allows two clauses with the same variable but opposite literal phases to be deselected simultaneously. It thus increases the deselection of clauses per try, and strengthens the learning in existential quantification levels. It turns out that the cube distribution concept can also be applied to SSAT solving under the clause deselection framework. Our SSAT algorithm is implemented based on CUED.

Maximal Clause Pruning: Consider the SSAT formula Φ in Eq. (2). As discussed in Section 6.1, if a clause C_i is locally deselected by $\tau_1 \in X_1$ and $\tau_2 \in X_2$ and $\mathsf{Pr}[\Phi|_{\tau_1}] = \mathsf{Pr}[\Phi|_{\tau_1\tau_2}]$, the selection literal $\neg t_i^1$ can be discarded from the selection minterm $m_{T_1} = \psi_1|_{\tau_1}$. However, the removal of such selection literals may not be maximal. If there exists an assignment $\tau_2'(X_2)$ such that it preserves the selection status of clauses in $\phi|_{\tau_1}$, i.e. $\phi|_{\tau_1\tau_2} = \phi|_{\tau_1\tau_2'}$, and apart from the already deselected ones, locally deselects clauses $C_i \in \phi \setminus \phi|_{\tau_1}$ where $\neg t_i^1 \in m_{T_1}$, $\neg t_i^1$ can be discarded from m_{T_1} and a stronger learnt clause can be obtained. The subroutine *MaximalPruning* in Algorithm 2 accomplishes this by solving ψ_2 under the assumption that at least one such clause should be deselected while the selection status of clauses in $\phi|_{\tau_1}$ is preserved.

Non-chronological Backtracking: Consider the SSAT formula Φ in Eq. (1). According to Sections 5 and 6, if a subproblem $\Phi|_{\tau_1}$ where $\tau_1(X_1)$ has satisfying probability equal to 0, a learnt clause is added to enforce the deselection of at least one of the selected clauses at the first level to exclude the subproblems unworthy of trying. However, if the deselection is impossible at the current quantification level, say, level k, it must be done at a lower level; that is, the parent problems of Φ. By finding the maximum quantification level, also known as the *backtrack level* (*btlev*), which the deselection is possible, a learnt clause can be added at that level and the solving process may continue from there.[1] In Algorithms 1 and 2, *AddLearntClausesToPriorLevels* adds learnt clauses to ψ_j where *btlev* $\leq j < k$ if the deselection is impossible at current level k.

Example 4. *Consider the SSAT formula* Φ *and its selection relations* ψ_1 *and* ψ_2 *at the first and second levels, respectively, with*

$$\begin{aligned}
\Phi &= \exists x, \text{Я}^{0.4} y, \exists z_1, z_2.(x)(\overline{x} \vee y \vee z_1)(\overline{x} \vee z_2)(\overline{x} \vee \overline{z_2}),\\
\psi_1 &= (t_1^1 \equiv \neg x)(t_2^1 \equiv x)(t_3^1 \equiv x)(t_4^1 \equiv x),\\
\psi_2 &= (t_1^2 \equiv \top)(t_2^2 \equiv \neg y)(t_3^2 \equiv \top)(t_4^2 \equiv \top).
\end{aligned}$$

Let the tried assignments be $\tau_1 = x$, $\tau_2 = \neg y$, *and* $\tau_3 = z_1 z_2$, *which deselect* C_1, C_2, *and* C_3. *A conflict caused by* C_3 *and* C_4 *can be detected at level 3. Because the deselection of* C_3 *and* C_4 *is impossible at levels 2 and 3, with non-chronological backtracking, we add a learnt clause* $(\neg t_3^1 \vee \neg t_4^1)$ *to* ψ_1 *at level 1, and get* $\mathsf{Pr}[\Phi|_{\tau_1}] = 0$. *After trying* $\tau_4 = \neg x$, *which falsifies* C_1, *we get* $\mathsf{Pr}[\Phi|_{\tau_4}] = 0$, *add* $(\neg t_1^1)$ *to* ψ_1, *and find* $\mathsf{SAT}(\psi_1) = \bot$. *We thus obtain* $\mathsf{Pr}[\Phi] = 0$.

In contrast, without non-chronological backtracking, we will return to level 2 and explore $\tau_5 = y$ *(an additional SAT call), find the conflict at level 3 (an additional SAT call), and add a learnt clause* $(\neg t_3^2 \vee \neg t_4^2)$ *to* ψ_2 *to deselect* C_3 *or* C_4 *at level 2. After finding* $\mathsf{SAT}(\psi_2) = \bot$ *(an additional SAT call), we add a learnt clause* $(\neg t_2^1 \vee \neg t_3^1 \vee \neg t_4^1)$ *to* ψ_1, *and get* $\tau_6 = \tau_4 = \neg x$. *This process requires three additional SAT calls to learn the root cause of the conflict at level 1.*

8 Bounds for Incomplete SSAT

The proposed algorithm can be easily modified to provide upper and/or lower bounds on the satisfying probability under computation in case the solving cannot be completed in time. Since the proposed algorithm considers all variables at each level simultaneously, the intermediate information is valid and useful for deriving bounds to the exact satisfying probability. Depending on the quantification type Q_1 of the first level, the bounds can be computed as follows.

For $Q_1 = \exists$, the encountered largest satisfying probability of subproblems serves as a lower bound. On the other hand, since we cannot tell whether there exists an assignment $\tau(X_1)$ letting $\mathsf{Pr}[\Phi|_\tau] = 1$ until $\mathsf{SAT}(\psi_1) = \bot$, the upper bound 1 cannot be reduced. However, for an SSAT formula whose matrix negation is available, the upper bound can be tightened by solving the lower bound of the formula that is same as the original one but with the matrix being negated. For a CNF formula converted from a circuit by Tseitin transformation, its negation can be obtained easily.

For $Q_1 = \text{Я}$, from Section 6.2, since the collected selection cubes characterize the searched space and are valid throughout the solving process, the lower bound *LB* can be computed by Eq. (5). The upper bound *UB* can be ob-

[1] In our recursive implementation, if the deselection of clauses is impossible at current level k (no literals of level k exist in the clauses), a learnt clause is added only at level $k-1$ to enforce the deselection at level $k-1$. If the deselection is still impossible at level $k-1$ (no literals of level $k-1$ exist in the clauses), another learnt clause is added at level $k-2$. The backtrack process continues until the deselection is found possible at *btlev* $< k$. Note that the above visits to levels greater than *btlev* are purely due to our recursive implementation and involves only simple syntactic checking whether the learnt clause has a literal of the current quantification level. The backtrack is essentially non-chronological.

tained by treating the satisfying probabilities of the unexplored subproblems as 1, which can be expressed as

$$UB = LB + 1 \times (1 - \sum_{(p,S_p)\in V} \mathsf{Pr}[\text{Ɐ} X_1.\varphi]),$$

where p, S_p, V, and φ are the same as those defined in Theorem 1.

9 Experimental Results

The proposed clause-selection-based algorithm, named `ClauSSat`, was implemented[2] in the C++ language under the QBF framework of `CUED` (Chen and Jiang 2019). `Glucose-4.1` (Audemard and Simon 2009), which is based on `Minisat-2.2` (Eén and Sörensson 2003), and `Cachet` (Sang et al. 2004; Sang, Beame, and Kautz 2005) were adopted as the underlying SAT and model counting engines, respectively.[3] All experiments were conducted on a Linux machine with Intel Core i7-8700 CPU of 3.2 GHz and 32 GB RAM. A time limit of 1000 seconds was imposed on solving an instance in the experiments. No memory limitation was imposed, but the maximum memory usage during execution was recorded.

We compared `ClauSSat` with the state-of-the-art multi-level SSAT solver `DC-SSAT` (Majercik and Boots 2005), and the two-level solvers `erSSAT` (Lee, Wang, and Jiang 2018) and `reSSAT` (Lee, Wang, and Jiang 2017), both of which use `Minisat-2.2` as the underlying SAT engine. We note that the performance of `ClauSSat` was little affected by the choice of engines `Minisat-2.2` and `Glucose-4.1` in our experiments. The solvers were evaluated on 23 families of 318 SSAT formulas in total. Among them, 13 families consist of multi-level formulas and 10 consist of two-level ones. Due to space limit, we only reported the results of 16 families, each with up to 3 sampled formulas, in Table 1. Those not included are mostly either easy or hard for all the solvers compared. In the table, the first 9 families are multi-level formulas converted from QBF instances on QBFLIB (Giunchiglia, Narizzano, and Tacchella 2001) by substituting randomized quantifiers for universal quantifiers with probabilities p randomly chosen $\in [0, 1]$. The next 5 and last 2 families are exist-random and random-exist quantified SSAT formulas used in (Lee, Wang, and Jiang 2018) and (Lee, Wang, and Jiang 2017), respectively. In particular, Families 1-4 include formulas that encode planning problems; Families 5-7 encode verification problems; Families 8-9 encode modal logic formulas (Pan and Vardi 2003); Families 10-12 encode conformant planning problems; Family 13 encode the *quantitative information flow* (QIF) problem (Fremont, Rabe, and Seshia 2017); Families 14-15 encode the *probabilistic equivalence checking* problem (Lee and Jiang 2018); Family 16 encode the *strategic companies* problem (Cadoli, Eiter, and Gottlob 1997).

9.1 Comparison to State-of-the-Art Solvers

In the experiments, `ClauSSat` is evaluated with all 3 enhancement techniques of Section 7 enabled. The results are shown in Table 1, where Columns 2-5 report the prefix, the numbers of existentially and randomly quantified variables (#$V_\exists$ and #$V_{\text{Ɐ}}$, respectively), and the number of clauses (#C) of each benchmark. In the second column, the notation Σ_i (resp. Π_i) indicates that the prefix starts with an existential (resp. randomized) quantifier and has i quantification levels in total. For `ClauSSat`, `erSSAT`, and `reSSAT`, the time (T_1) spent to reach the lower bound (LB) and the entire runtime (T_2) are reported. If the solver fails to give exact answers before timeout, T_2 will be left as "-". Also, for the formulas where $Q_1 = \text{Ɐ}$, since `ClauSSat` and `reSSAT` gives lower and upper bounds (UB) at the end of the program, T_1 is equal to T_2. If no bounds are solved, all entries are left as "-". `DC-SSAT`, as an exact solver, either exactly solves the formula (reporting satisfying probability (Pr) and runtime (T)) or timeouts (both left as "-"). While the results of `ClauSSat` and `DC-SSAT` are shown in Columns 6-9 and 10-11, respectively, those of `erSSAT` and `reSSAT` are shown jointly in Columns 12-15 without ambiguity due to their distinct applicability on the formulas.

As can be seen, the results show that `ClauSSat` outperforms the others in most of the families. Specifically, for the QBF converted SSAT formulas, `ClauSSat` exactly solved or derived tightest lower bounds, while `DC-SSAT` failed to solve most of the cases. For Adder and k_ph_p, `ClauSSat` derived lower bounds for more formulas than `DC-SSAT`. For ev-pr-4x4 and k_branch_n, `DC-SSAT` performed particularly well. These two families seem to be easy for search based solvers but not for clause-selected based solvers as evidenced by the fact that their original QBF counterparts can be efficiently solved by `DepQBF` (Lonsing and Egly 2017) but not by `CUED`. For E-MAJSAT families, including MPEC, Toilet-A, Conformant, and QIF, `ClauSSat` outperformed all the others in terms of the number of achieved tightest lower bounds. In particular, `ClauSSat` exactly solved the most cases in Toilet-A and reached the lower bounds achieved by `erSSAT` in shorter time. For Sand-Castle, `DC-SSAT` outperformed `ClauSSat` and `erSSAT` without much surprise because it is designed to solve such conformant planning problems, while `ClauSSat` still achieved lower bounds greater than 0.99. For PEC, `ClauSSat` derived reasonable bounds for all formulas. In contrast, `DC-SSAT` solved one and `reSSAT` failed to solve any. For stracomp, although `ClauSSat` took longer than `reSSAT`, both solvers outperformed `DC-SSAT` by exactly solving all the formulas. Besides the above comparison, we also experimented with the E-MAJSAT solver `MaxCount` (Fremont, Rabe, and Seshia 2017), which performed superior to all other solvers on program synthesis benchmarks, but inferior to `erSSAT` on planning benchmarks. As comparisons between `erSSAT` and `MaxCount` are available in (Lee, Wang, and Jiang 2018), we omitted showing the results of `MaxCount` from Table 1.

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

[3] We used `Cachet`, but not other more advanced model counters, in order to demonstrate that `ClauSSat` is superior to `erSSAT` (which provides `Cachet` and BDD options for model counting and uses BDD as its default option for better performance) not due to the Sharp-SAT improvement but due to the algorithmic advancement.

benchmark statisitcs					ClauSSat				DC-SSAT		{erSSAT, reSSAT}			
family/formula	pfx	$\#V_{\exists}$	$\#V_{\rotatebox[origin=c]{180}{R}}$	#C	LB	UB	T_1	T_2	Pr	T	LB	UB	T_1	T_2
Connect2														
3x5_w_	Σ_{17}	2956	24	9.1k	**2.78e-1**	**2.78e-1**	2	391	-	-	-	-	-	-
3x6_w_	Σ_{19}	4141	27	13.1k	**1.76e-1**	1	2	-	-	-	-	-	-	-
3x7_w_	Σ_{19}	5521	33	17.8k	**1.39e-1**	1	2	-	-	-	-	-	-	-
ev-pr-4x4														
5-3-0-0-1-lg	Σ_5	884	12	6.0k	**1**	**1**	1	1	**1**	0	-	-	-	-
7-3-0-0-1-lg	Σ_7	1233	18	8.3k	-	-	-	-	**1**	0	-	-	-	-
9-3-0-0-1-lg	Σ_9	1582	24	10.6k	-	-	-	-	**1**	0	-	-	-	-
pipesnotankage														
02_5	Σ_5	477	2	9.8k	**9.99e-1**	1	14	-	-	-	-	-	-	-
02_6	Σ_5	504	2	9.7k	**9.49e-1**	1	188	-	-	-	-	-	-	-
02_7	Σ_5	510	2	9.6k	**2.53e-1**	1	465	-	-	-	-	-	-	-
depots														
01_5	Σ_5	360	2	5.8k	**5.16e-1**	1	4	-	-	-	-	-	-	-
QBF-Hardness														
10-error01...-22	Π_{46}	5488	240	25.0k	**1.00e0**	1	-	-	-	-	-	-	-	-
10-error01...-23	Π_{48}	5749	250	26.6k	**1.00e0**	1	-	-	-	-	-	-	-	-
10-error01...-24	Π_{50}	6011	260	28.2k	**2.61e-1**	1	-	-	-	-	-	-	-	-
Counter														
03	Σ_5	77	3	0.2k	**9.80e-1**	1	56	-	-	-	-	-	-	-
03e	Σ_5	88	3	0.2k	**9.94e-1**	1	6	-	-	-	-	-	-	-
03r	Σ_5	88	3	0.2k	**9.93e-1**	1	7	-	-	-	-	-	-	-
Adder														
Adder2-2-c	Σ_7	224	12	0.3k	**8.82e-1**	**8.82e-1**	42	174	**8.82e-1**	0	-	-	-	-
adder-2-unsat	Σ_3	44	9	0.1k	**1.00e0**	**1.00e0**	19	19	**1.00e0**	1	-	-	-	-
adder-4-unsat	Σ_3	194	38	0.5k	**1.00e0**	**1.00e0**	3	3	-	-	-	-	-	-
k_branch_n														
3	Σ_{11}	502	13	1.5k	-	-	-	-	**1**	0	-	-	-	-
5	Σ_{15}	1124	25	3.9k	-	-	-	-	**1**	0	-	-	-	-
7	Σ_{19}	1994	33	7.5k	-	-	-	-	**1**	0	-	-	-	-
k_ph_p														
2	Σ_5	42	2	0.1k	**9.25e-1**	**9.25e-1**	6	6	**9.25e-1**	0	-	-	-	-
3	Σ_5	95	4	0.3k	**1.00e0**	**1.00e0**	1	1	**1.00e0**	1	-	-	-	-
4	Σ_5	175	5	0.6k	**9.68e-1**	1	392	-	-	-	-	-	-	-
ToiletA														
10_05.3	Σ_3	240	10	12.0k	**3.13e-2**	1	4	-	-	-	1.56e-2	1	0	-
10_05.4	Σ_3	320	10	12.7k	**1.25e-1**	1	827	-	-	-	1.56e-2	1	182	-
10_10.2	Σ_3	160	10	11.3k	**1**	**1**	673	673	-	-	**1**	**1**	2	2
conformant														
cube_c9...-11_	Σ_3	918	10	24.5k	**3.12e-1**	1	724	-	-	-	2.89e-1	1	111	-
emptyroom...-22_	Σ_3	2144	8	20.2k	**5.47e-2**	1	178	-	-	-	3.91e-3	1	134	-
emptyroom...-44_	Σ_3	4256	8	32.4k	**1.25e-1**	1	797	-	-	-	3.91e-3	1	247	-
Sand-Castle														
SC-15	Σ_3	62	75	0.3k	9.93e-1	1	135	-	**9.94e-1**	0	**9.94e-1**	1	692	-
SC-16	Σ_3	66	80	0.3k	9.95e-1	1	687	-	**9.96e-1**	1	9.95e-1	1	990	-
SC-17	Σ_3	70	85	0.3k	9.96e-1	1	618	-	**9.97e-1**	3	9.95e-1	1	898	-
QIF														
bin-search-16	Σ_3	1432	16	5.8k	**1.04e-2**	1	837	-	-	-	1.95e-3	1	62	-
CVE-2007-2875	Σ_3	752	32	1.7k	**1**	**1**	0	0	-	-	**1**	**1**	1	1
reverse	Σ_3	197	32	0.3k	**3.40e-5**	1	610	-	-	-	5.96e-7	1	471	-
MPEC														
c880-er	Σ_3	449	2	1.2k	**2.34e-1**	1	76	-	-	-	1.25e-1	1	0	-
c5315-er	Σ_3	908	10	2.2k	3.30e-1	1	376	-	-	-	**4.14e-1**	1	71	-
c7552-er	Σ_3	643	5	1.3k	**4.87e-1**	1	0	-	-	-	2.34e-1	1	0	-
PEC														
c432_re	Π_2	297	33	0.9k	**1.67e-2**	**1.67e-2**	99	99	-	-	-	-	-	-
c880_re	Π_2	396	55	1.2k	**4.00e-6**	**8.26e-2**	-	-	-	-	-	-	-	-
c1908_re	Π_2	239	31	0.7k	1.07e-4	7.54e-4	-	-	**7.45e-4**	66	-	-	-	-
stracomp														
x75.4	Π_2	2254	75	7.0k	**1**	**1**	576	576	-	-	**1**	**1**	111	111
x75.9	Π_2	2254	75	7.0k	**1**	**1**	237	237	-	-	**1**	**1**	158	158
x75.19	Π_2	2254	75	7.0k	**1**	**1**	310	310	-	-	**1**	**1**	121	121
Maximum memory usage (GB)								2.4		32.1				3.1

Table 1: Results for solver performance comparison.

By examining the instances, e.g. the family Sand-Castle, on which `DC-SSAT` performs better than `ClauSSat`, we observed that they exhibit the decomposability exploited by `DC-SSAT`. `ClauSSat` currently does not exploit such a decomposition strategy, but the applicability of such a divide-and-conquer approach can be further studied.

In summary, among the whole collection of 318 formulas, `ClauSSat` exactly solved 188 and derived tightest bounds for 98 while `DC-SSAT` exactly solved 169. On the other hand, among the 215 two-level SSAT instances, `ClauSSat` (resp. `erSSAT` and `reSSAT` combined) exactly solved 127 (resp. 119) and derived tightest bounds for 71 (resp. 23). Also, for maximum memory usage, `ClauSSat` consumes memory an order of magnitude less than that of `DC-SSAT`, and is comparable to that of `erSSAT` and `reSSAT`. The results suggest the advancement of `ClauSSat` over the state-of-the-art.

9.2 Evaluation of Enhancement Techniques

To investigate the efficacy of the enhancement techniques, we ran `ClauSSat` under different settings. Let the enabled enhancement techniques be referred to as `c` for cube distribution, `m` for maximal clause pruning, and `b` for non-chronological backtracking. `ClauSSat`-$\{$`mc`$\}$ exactly solved 11 more formulas and provided tighter bounds for 23 more formulas than `ClauSSat`-$\{$`m`$\}$. Further, `ClauSSat`-$\{$`mcb`$\}$ exactly solved 18 more formulas and provided tighter bounds for 34 more formulas compared to `ClauSSat`-$\{$`mc`$\}$. The statistics reveal the effectiveness of the enhancement techniques.

10 Conclusions and Future Work

We have lifted the clause-selection framework of QBF solving to the SSAT domain. A new SSAT solver `ClauSSat` has been developed and strengthened. Experiments have demonstrated the superiority of our solver compared to other state-of-the-art solvers on various application formulas. For future work, as approximate model counting gains recent advancements (Soos, Gocht, and Meel 2020), we would like to study its applicability in our SSAT solving framework. Also we would like to generalize our solving techniques to dependency SSAT (DSSAT) (Lee and Jiang 2021).

Acknowledgments

This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant No. 108-2221-E-002-144-MY3 and 108-2218-E-002-073. JHJ was supported in part by the Alexander von Humboldt Foundation.

References

Audemard, G.; and Simon, L. 2009. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 399–404.

Cadoli, M.; Eiter, T.; and Gottlob, G. 1997. Default Logic as a Query Language. *IEEE Transactions on Knowledge and Data Engineering* 9(3): 448–463.

Chakraborty, S.; Meel, K. S.; and Vardi, M. Y. 2016. Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 3569–3576.

Chen, L.-C.; and Jiang, J.-H. R. 2019. A Cube Distribution Approach to QBF Solving and Certificate Minimization. In *Principles and Practice of Constraint Programming*, 529–546.

Eén, N.; and Sörensson, N. 2003. An Extensible SAT-solver. In *Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 502–518.

Fremont, D. J.; Rabe, M. N.; and Seshia, S. A. 2017. Maximum Model Counting. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 3885–3892.

Freudenthal, E.; and Karamcheti, V. 2003. QTM: Trust Management with Quantified Stochastic Attributes. NYU Computer Science Technical Report TR 2003-848.

Giunchiglia, E.; Narizzano, M.; and Tacchella, A. 2001. Quantified Boolean Formulas Satisfiability Library (QBFLIB). http://www.qbflib.org (last accessed in March 2020).

Gomes, C. P.; Hoffmann, J.; Sabharwal, A.; and Selman, B. 2007. From Sampling to Model Counting. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2007, 2293–2299.

Gomes, C. P.; Sabharwal, A.; and Selman, B. 2006. Model Counting: A New Strategy for Obtaining Good Bounds. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 54–61.

Janota, M.; and Marques-Silva, J. 2015. Solving QBF by Clause Selection. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 325–331.

Lee, N.-Z.; and Jiang, J.-H. R. 2018. Towards Formal Evaluation and Verification of Probabilistic Design. *IEEE Transactions on Computers* 67(8): 1202–1216.

Lee, N.-Z.; and Jiang, J.-H. R. 2021. Dependency Stochastic Boolean Satisfiability: A Logical Formalism for NEXPTIME Decision Problems with Uncertainty. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*.

Lee, N.-Z.; Wang, Y.-S.; and Jiang, J.-H. R. 2017. Solving Stochastic Boolean Satisfiability under Random-Exist Quantification. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 688–694.

Lee, N.-Z.; Wang, Y.-S.; and Jiang, J.-H. R. 2018. Solving Exist-Random Quantified Stochastic Boolean Satisfiability via Clause Selection. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 1339–1345.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The Computational Complexity of Probabilistic Planning. *Journal of Artificial Intelligence Research* 9: 1–36.

Littman, M. L.; Majercik, S. M.; and Pitassi, T. 2001. Stochastic Boolean Satisfiability. *Journal of Automated Reasoning* 27(3): 251–296.

Lonsing, F.; and Egly, U. 2017. DepQBF 6.0: A Search-based QBF Solver Beyond Traditional QCDCL. In *Proceedings of International Conference on Automated Deduction (CADE)*, 371–384.

Majercik, S. M. 2009. Stochastic Boolean Satisfiability. In *Handbook of Satisfiability*, 887–925. IOS Press.

Majercik, S. M.; and Boots, B. 2005. DC-SSAT: A Divide-and-Conquer Approach to Solving Stochastic Satisfiability Problems Efficiently. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 416–422.

Pan, G.; and Vardi, M. Y. 2003. Optimizing a BDD-based Modal Solver. In *Proceedings of International Conference on Automated Deduction (CADE)*, 75–89.

Papadimitriou, C. 1985. Games Against Nature. *Journal of Computer and System Sciences* 31(2): 288–301.

Sang, T.; Bacchus, F.; Beame, P.; Kautz, H. A.; and Pitassi, T. 2004. Combining Component Caching and Clause Learning for Effective Model Counting. In *Proceedings of International Conference on Theory and Applications of Satisfiability Testing (SAT)*.

Sang, T.; Beame, P.; and Kautz, H. A. 2005. Performing Bayesian Inference by Weighted Model Counting. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, volume 5, 475–481.

Soos, M.; Gocht, S.; and Meel, K. S. 2020. Tinted, Detached, and Lazy CNF-XOR Solving and Its Applications to Counting and Sampling. In *Proceedings of International Conference on Computer Aided Verification (CAV)*, 463–484.

Toda, S. 1991. PP Is as Hard as the Polynomial-Time Hierarchy. *SIAM Journal on Computing* 20(5): 865–877.