Solving Exist-Random Quantified Stochastic Boolean Satisfiability via Clause Selection

Nian-Ze Lee[*1], **Yen-Shi Wang**[*2] and **Jie-Hong R. Jiang**[1,2]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
{d04943019, b03901116, jhjiang}@ntu.edu.tw

Abstract

Stochastic Boolean satisfiability (SSAT) is an expressive language to formulate decision problems with randomness. Solving SSAT formulas has the same PSPACE-complete computational complexity as solving quantified Boolean formulas (QBFs). Despite its broad applications and profound theoretical values, SSAT has received relatively little attention compared to QBF. In this paper, we focus on exist-random quantified SSAT formulas, also known as E-MAJSAT, which is a special fragment of SSAT commonly applied in probabilistic conformant planning, posteriori hypothesis, and maximum expected utility. Based on clause selection, a recently proposed QBF technique, we propose an algorithm to solve E-MAJSAT. Moreover, our method can provide an approximate solution to E-MAJSAT with a lower bound when an exact answer is too expensive to compute. Experiments show that the proposed algorithm achieves significant performance gains and memory savings over the state-of-the-art SSAT solvers on a number of benchmark formulas, and provides useful lower bounds for cases where prior methods fail to compute exact answers.

1 Introduction

Stochastic Boolean satisfiability (SSAT) was first formulated in [Papadimitriou, 1985] and interpreted as *games against nature*. In an SSAT formula, a Boolean variable can be *randomly quantified with probability p* by a *randomized quantifier*, which specifies that the variable evaluates to TRUE with probability p. With randomized quantifiers, SSAT serves as a natural formalism for an abundance of computational problems endowed with randomness, such as propositional probabilistic planning, trust management, and Bayesian network inference [Hnich *et al.*, 2011; Littman *et al.*, 2001; Majercik, 2009]. The verification problem of VLSI circuits with probabilistic errors has also been investigated under the framework of SSAT recently [Lee and Jiang, 2014]. From the perspective of computational complexity, SSAT lies in the complexity class PSPACE-complete, the same as quantified Boolean formula (QBF). Therefore, advancing the scalability of SSAT solving not only benefits practical applications, but also has profound theoretical values.

In this paper, we focus on the exist-random quantified SSAT formulas of the form $\Phi = \exists X \mathfrak{R} Y.\phi$, which is known as *E-MAJSAT* [Littman *et al.*, 1998]. Computational problems, such as computing a *maximum a posteriori* (MAP) hypothesis or a *maximum expected utility* (MEU) solution [Dechter, 1998] in belief networks, and searching an optimal plan for probabilistic conformant planning domains [Littman *et al.*, 1998], can be formulated with E-MAJSAT.

Among previous endeavors, several techniques have been investigated to solve SSAT. Prior work `MAXPLAN` [Majercik and Littman, 1998] utilizes Davis-Putnam-Logemann-Loveland (DPLL) search [Davis *et al.*, 1962] and improves its efficiency by considering pure variables, unit propagation, and subproblem memorization. `DCSSAT` [Majercik and Boots, 2005] enhances `MAXPLAN` by applying the divide-and-conquer strategy to break SSAT formulas into many subproblems and solve them separately. The technique of *knowledge compilation* has also been exploited to solve E-MAJSAT. `ComPlan` [Huang, 2006] compiles the problems into *deterministic, decomposable negation normal form* (d-DNNF) [Darwiche, 2001; 2002], and performs a branch-and-bound search. It is further improved in [Pipatsrisawat and Darwiche, 2009] by an enhanced bound computation method. Recent work [Fremont *et al.*, 2017] proposed a *maximum model counting* algorithm for solving a special case of E-MAJSAT with all probabilities of randomized quantifiers equal to 0.5.

Recently, modern SAT techniques have been utilized to solve random-exist quantified SSAT formulas in [Lee *et al.*, 2017], which inspires us to examine the possibility of applying QBF solving techniques in the SSAT domain.

In this paper, we exploit the *clause selection* technique, which has been introduced in QBF solving recently [Janota and Marques-Silva, 2015; Rabe and Tentrup, 2015], and propose a *clause containment learning* method for E-MAJSAT solving. To the best of our knowledge, this paper is the first attempt to adopt approaches developed for QBF to solve SSAT. In addition to the learning technique, weighted model counting, which has been widely adopted in probabilistic inference [Chavira and Darwiche, 2008; Sang *et al.*, 2005], is

[*]The authors contributed equally to this work.

utilized in the proposed algorithm to compute probabilities.

We evaluate the proposed algorithm with a wide range of benchmarks including random formulas, planning problems, and probabilistic circuit verification. Experiments show that our method achieves significant performance gains and memory savings compared to the state-of-the-art SSAT solvers.

2 Preliminaries

We represent Boolean values TRUE and FALSE by symbols $\top$ and $\bot$, respectively. In the sequel, a variable x is assumed in the Boolean domain $\mathbb{B} = \{\top, \bot\}$. A *literal* is a variable (called a *positive literal*) or the negation of a variable (called a *negative literal*). For a literal l, let $\texttt{var}(l)$ denote the variable of l. Boolean connectives $\neg, \vee, \wedge, \Rightarrow, \equiv$ are interpreted in their conventional meanings. A *clause* is a disjunction of literals. A propositional Boolean formula ϕ is in *Conjunctive Normal Form* (CNF) if ϕ is a conjunction of clauses. A variable x is said to be *pure* in a formula if its appearances in the formula are all in the *positive phase* x or in the *negative phase* $\neg x$. A *cube* is a conjunction of literals. In the sequel, we assume propositional Boolean formulas are in CNF.

A Boolean formula ϕ over a set of variables $X = \{x_1, \ldots, x_n\}$ defines a unique Boolean function $\mathbb{B}^n \to \mathbb{B}$. Let $\texttt{vars}(\phi)$ denote the set of variables appearing in a Boolean formula ϕ. An *assignment* τ over a set of variables $X \subseteq \texttt{vars}(\phi)$ for a formula ϕ is a mapping $\tau : X \to \mathbb{B}$. An assignment $\tau : X \to \mathbb{B}$ is a *complete assignment* for formula ϕ if $X = \texttt{vars}(\phi)$; otherwise, i.e., $X \subset \texttt{vars}(\phi)$, it is a *partial assignment*. Given a Boolean formula ϕ and an assignment τ over $\texttt{vars}(\phi)$, the *cofactor* of ϕ under τ, denoted by $\phi|_\tau$, is derived by substituting every occurrence of each variable $x \in \texttt{vars}(\phi)$ in ϕ by $\tau(x)$. If $\phi|_\tau = \top$, we call τ a satisfying assignment of ϕ. The satisfiability problem of a Boolean formula ϕ asks whether or not ϕ has a satisfying assignment. We write $\texttt{SAT}(\phi) = \top$ to denote that ϕ is satisfiable. A satisfying assignment of ϕ is also called a *model* of ϕ. On the other hand, if ϕ has no satisfying assignment, it is unsatisfiable and written as $\texttt{SAT}(\phi) = \bot$. Given two Boolean formulas ϕ and ψ, we write $\phi \models \psi$ if every satisfying assignment for ϕ also satisfies ψ. In the sequel, we alternatively represent an assignment τ as a cube, a clause C as a set of literals, and a CNF formula as a set of clauses.

2.1 Stochastic Boolean Satisfiability

A *stochastic Boolean satisfiability* (SSAT) formula is of the form

$$\Phi = Q_1 x_1 \ldots Q_n x_n . \phi(x_1, \ldots, x_n),$$

where $Q_i \in \{\exists, \mathrm{R}^{p_i}\}$ and ϕ is a quantifier-free Boolean formula. Symbol $\exists$ denotes the existential quantifier, and R^{p_i} denotes the randomized quantifier on x_i for the probability of $x_i = \top$ equal to $p_i \in [0, 1]$. Given an SSAT formula Φ, the quantification structure $Q_1 x_1 \ldots Q_n x_n$ is called the *prefix*, and the Boolean formula ϕ is called the *matrix*.

Let v be the outermost variable in the prefix of an SSAT formula Φ. The satisfying probability of Φ, denoted by $\Pr[\Phi]$, can be computed recursively by the following rules.

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

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

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

d) $\Pr[\Phi] = (1-p)\Pr[\Phi|_{\neg v}] + p \Pr[\Phi|_v]$, if v is randomly quantified by R^p.

In this paper, we focus on solving *E-MAJSAT*, the exist-random fragment of SSAT with the form $\Phi = \exists X \mathrm{R} Y . \phi(X, Y)$, where X and Y are two disjoint sets of variables. The satisfying probability of Φ is obtained through maximizing $\Pr[\Phi|_\tau]$, called the satisfying probability of Φ conditioned on τ, over all assignments τ on X.

2.2 Model Counting

The *model counting* problem of a Boolean formula ϕ asks to find the number of satisfying assignments of ϕ. Model counting algorithms can be classified into two categories: *exact model counting* and *approximate model counting*. The former computes the exact number of satisfying assignments of a formula; the latter computes an upper or a lower bound of the number of satisfying assignments with some confidence level. In its weighted version, a weight function ω is defined to map each variable $x \in \texttt{vars}(\phi)$ to some *weight* $\omega(x) \in [0, 1]$, which can be seen as the probability $\Pr[x = \top]$. The weight of a positive literal x (resp. negative literal $\neg x$) of variable x is defined to be $\omega(x)$ (resp. $1 - \omega(x)$). The weight of an assignment is defined to be the product of the weights of its individual literals. The weight of a formula equals the summation of weights of its satisfying assignments.

Given an E-MAJSAT formula $\Phi = \exists X \mathrm{R} Y . \phi(X, Y)$ and an assignment τ on X, cofactoring the matrix with τ results in a formula $\phi|_\tau$ referring only to variables in Y. The prefix $\mathrm{R}Y$ induces a weight function $\omega : Y \to [0, 1]$ 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 $\Pr[\Phi|_\tau]$, which equals the weight of the formula $\phi|_\tau$ under the weight function ω, can be obtained by invoking a weighted model counter on the formula $\phi|_\tau$ with the weight function ω. In the sequel, the invocation of a weighted model counter is expressed by $\texttt{WeightModelCount}(\mathrm{R}Y . \phi|_\tau)$, which returns the conditional satisfying probability $\Pr[\mathrm{R}Y . \phi|_\tau]$.

2.3 Clause Selection

Clause selection is a recently proposed technique for QBF solving [Janota and Marques-Silva, 2015; Rabe and Tentrup, 2015]. Given a CNF formula $\phi(X, Y)$ over a set of variables $X \cup Y$ with $X \cap Y = \emptyset$, 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 in X (resp. Y). For example, for $C = (x_1 \vee x_2 \vee y_1 \vee y_2)$, 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 every literal in C^X is assigned to $\bot$ by τ; C is said to be *deselected* by τ if some literal in C^X is assigned to $\top$ by τ; C is said to be *undecided* if it is neither selected nor deselected. We also use $\phi|_\tau$ to denote the set of clauses selected by the assignment τ. 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 the *selection relation* of $\phi(X, Y)$.

Example 1 *Consider a CNF formula $\phi(X, Y)$ over two sets of variables $X = \{e_1, e_2, e_3\}$ 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 selection variable s_i is introduced for each clause, and $S = \{s_1, s_2, s_3, s_4\}$. 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 the complete assignment $\tau_1 = \neg e_1 \neg e_2 \neg e_3$ over X. It selects C_1 and C_2, and deselects C_3 and C_4, as can be seen from the selection relation cofactored by τ_1, which results in $\psi(X, S)|_{\tau_1} = s_1 s_2 \neg s_3 \neg s_4$. Consider the partial assignment $\tau_2 = \neg e_1 e_3$ 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.

3 Clause Containment Learning

Consider an E-MAJSAT formula $\Phi = \exists X \text{MAJ} Y.\phi$. To obtain the satisfying probability of Φ, it suffices to enumerate every assignment τ on X, and calculate the corresponding conditional satisfying probability $\Pr[\Phi|_\tau]$. Clearly, the above brute-force approach is computationally expensive. Inspired 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 on 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.

Property 1 *Given an E-MAJSAT formula $\Phi = \exists X \text{MAJ} Y.\phi(X, Y)$ and two assignments τ_1 and τ_2 over X, we have*

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

By Property 1 and clause selection, we propose clause containment learning as detailed below. 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 every clause in $\phi|_{\tau_1}$, 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 by Property 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 idea in the proposed clause containment learning is to block every unexplored assignment τ_2 that selects a set of clauses $\phi|_{\tau_2}$ containing a set of clauses $\phi|_{\tau_1}$ previously 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 $\bot$ if and only if C is deselected. Therefore, a learnt clause, which is the disjunction of the negation of selection variables of clauses in $\phi|_{\tau_1}$, is deduced to record this information. The above idea gives rise to the proposed algorithm in Figure 1 to solve E-MAJSAT formulas. (Lines `03`, `07`, `08` and `11` describe enhancement techniques of the proposed algorithm, which will be discussed in Section 4.)

SolveEMAJSAT
input: $\Phi = \exists X \text{MAJ} Y.\phi(X, Y)$
output: $\Pr[\Phi]$
begin
01 $\psi(X, S) := (\bigwedge_{C\in\phi}(s_C \equiv \neg C^X)) \wedge (\bigwedge_{\text{pure } l:\text{var}(l)\in X} l)$;
02 `prob` := 0;
03 `s-table` := `BuildSubsumeTable`(ϕ);
04 **while** `SAT`(ψ) $= \top$
05 τ := the found model of ψ for variables in X;
06 **if** `SAT`($\phi|_\tau$) $= \top$
07 τ' := `SelectMinimalClauses`(ϕ, ψ);
08 φ := `RemoveSubsumedClauses`($\phi|_{\tau'}$, `s-table`);
09 `prob` := max{`prob`, `WeightModelCount`(MAJ$Y.\varphi$)};
10 $C_S := \bigvee_{C\in\varphi} \neg s_C$;
11 C_L := `DiscardLiterals`(ϕ, C_S, `prob`);
12 **else** // `SAT`($\phi|_\tau$) $= \bot$
13 C_L := `MinimalConflicting`(ϕ, τ);
14 $\psi := \psi \wedge C_L$;
15 **return** `prob`;
end

Figure 1: E-MAJSAT solving with clause containment learning.

The algorithm involves 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, line `01` initializes the selection relation together with asserting the literals of pure X variables. 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 less clauses are selected, we can always assert pure X variables.

The selection relation is used to select different assignments τ over X. If $\phi|_\tau$ is satisfiable, a weighted model counter is invoked to compute the conditional satisfying probability $\Pr[\text{MAJ}Y.\phi|_\tau]$ under the assignment τ. The blocking clause C_L derived based on the proposed containment learning technique is 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 (e.g., using the subroutine `analyzeFinal` in `Minisat` [Eén and

Sörensson, 2003a; 2003b]). 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 probability of the input E-MAJSAT formula.

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

Example 2 *Continuing Example 1, we show how algorithm* `SolveEMAJSAT` *in Figure 1 (without the enhancement techniques) solves the E-MAJSAT instance* $\Phi = \exists e_1 e_2 e_3 \text{ᴙ}^{0.5} r_1 r_2 r_3 . \phi(X, Y)$. *Let the first tried assignment* τ_1 *be* $\neg e_1 \neg e_2 \neg e_3$, *which selects* C_1, C_2. *The algorithm derives* $\Pr[\text{ᴙ}^{0.5} Y . \phi|_{\tau_1}] = 0.75$ *by invoking the weighted model counter in line 09. 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 tried assignment* τ_2 *is* $\neg e_1 e_2 \neg e_3$, *which selects* C_1. *The weighted model counter gives* $\Pr[\text{ᴙ}^{0.5} Y . \phi|_{\tau_2}] = 0.75$, *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* $\Pr[\text{ᴙ}^{0.5} Y . \phi|_{\tau_3}] = 0.5$, *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* $\Pr[\text{ᴙ}^{0.5} Y . \phi|_{\tau_4}]$ *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, making* $\phi|_{\tau_5} = \top$ *and* $\Pr[\text{ᴙ}^{0.5} Y . \phi|_{\tau_5}] = 1$. *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.*

4 Enhancement Techniques

The computational efficiency of algorithm `SolveEMAJSAT` is greatly affected by the strength of 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 Figure 1, the enhancement techniques are executed by subroutines `SelectMinimalClauses` (in line `07`), `RemoveSubsumedClauses` (in line `08`), and `DiscardLiterals` (in line `11`), to be detailed in the following three sections, respectively.

4.1 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 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. The learnt clause is conjoined with $\psi(X, S)$ so that $\psi(X, S)$ is solved under the *unit assumption* [Eén and Sörensson, 2003b] to ensure that at least one of the currently selected clauses must be deselected in the future. The above process repeats until the selection relation becomes unsatisfiable.

4.2 Clause Subsumption

The second enhancement technique, *clause subsumption*, decreases the length of the 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. Using the lookup table of the subsumption relation computed by subroutine `BuildSubsumeTable`, the procedure `RemoveSubsumedClauses` simplifies the set of selected clauses $\phi|_{\tau}$ by removing subsumed clauses.

4.3 Partial Assignment Pruning

To illustrate the third enhancement technique *partial assignment pruning*, we first take a closer look at the learnt clause deduced by the proposed clause containment learning. Given a matrix $\phi(X, Y)$ and an assignment τ over X, a learnt clause consists of the negation of the selection variables of the selected clauses. For each 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 the disjunction of the sub-clauses C^X, i.e., $C_L = \bigvee_{C \in \phi|_{\tau}} C^X$.

From the above illustration, we observe that the learnt clause can be strengthened as follows. First, temporarily discard some literal l in the 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, literal l is discarded; otherwise, it is kept in the clause. Fourth, repeat the above steps for other literals.

Benefiting from the three enhancement techniques, the efficiency of the proposed algorithm is greatly improved, as will be shown in our experiments.

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 like DC−SSAT [Majercik and Boots, 2005], 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.

5 Experimental Results

A prototyping program[1] of the proposed algorithm SolveEMAJSAT was implemented in the ABC [Mishchenko, 2005] environment with SAT solver Minisat−2.2 [Eén and Sörensson, 2003a]. For weighted model counting, we tried Cachet [Sang *et al.*, 2004; 2005], but the overall performance was not satisfactory. Instead, we adopt the well-developed BDD package CUDD [Somenzi, 1998], and weighted model counting of the formula is done by traversing the BDD built from the formula, as proposed in [Darwiche and Marquis, 2002]. The experiments were conducted on a Linux machine with an Intel Xeon 2.1 GHz CPU and 126 GB RAM.

We evaluated the proposed algorithm on both random k-CNF and application formulas. We compared the proposed approach to the state-of-the-art SSAT solver DC−SSAT, the maximum model counter MAXCOUNT [Fremont *et al.*, 2017], which relies on the approximate counter APPROXMC [Chakraborty *et al.*, 2016], and the E-MAJSAT solver ComPlan. As ComPlan is not publicly available, we used the CNF-to-d-DNNF compilation time of c2d [Darwiche, 2002], a key step in ComPlan, to estimate the performance of ComPlan. Because MAXCOUNT can only handle randomized quantifiers with probability 0.5, to compare MAXCOUNT against other methods we convert formulas involving randomized quantifiers with probabilities not equal to 0.5 into equivalent ones with all probabilities equal to 0.5 by the conversion method proposed in [Lee and Jiang, 2014].

In the sequel, our proposed approach with all three enhancement techniques enabled, DC−SSAT, MAXCOUNT, and the CNF-to-d-DNNF compiler are referred to as erSSAT, Dc, Max, and c2d, respectively. A runtime limit of 1000 seconds was imposed on each formula, and the symbol "-" denotes timeout. We did not impose a memory limit, but recorded the maximum memory usage during execution.

5.1 Random k-CNF Formulas

The random k-CNF formulas were generated by the cnfgen package [Lauria, 2012]. A collection of 280 formulas were generated with k, i.e., the number of literals in a clause, ranging in $\{3, 4, 5, 6, 7, 8, 9\}$, the number of variables ranging in $\{10, 20, 30, 40, 50\}$, and clause-to-variable ratio ranging in $\{k-1, k, k+1, k+2\}$. Two formulas were generated for each parameter combination. To convert the propositional formulas into E-MAJSAT formulas, half of the variables are existentially quantified, and the rest are randomly quantified with probability 0.5.

[1] Available at https://github.com/nianzelee/ssatABC.git

Among the collection of 280 formulas, erSSAT solved 216 instances exactly and computed lower bounds for the rest 64 instances with confidence level 1, while Dc and c2d solved 210 and 209 instances, respectively. On the other hand, Max computed lower bounds for 257 instances with confidence level greater than 0.8. Moreover, the maximum memory usage of erSSAT is two orders (resp. one order) of magnitude less than that of Dc (resp. c2d). As the runtime and memory usage of ComPlan are bounded below by c2d, we conclude that erSSAT has the best performance among compared solvers over random formulas.

5.2 Application Formulas

We collected seven families of application formulas for evaluation. The first two families, Toilet-A and Conformant, are exist-forall-exist quantified QBFs from [Giunchiglia *et al.*, 2005]. We converted them into exist-random-exist SSAT formulas by changing universal quantifiers to randomized ones with probabilities 0.5. The third family Sand-Castle is a probabilistic conformant planning problem, which was encoded as an E-MAJSAT formula in [Majercik and Littman, 1998]. The fourth, fifth, and sixth families are taken from [Fremont *et al.*, 2017], which encode the *maximum satisfiability* (MaxSat), *quantitative information flow* (QIF), and *program synthesis* (PS) problems, respectively, into maximum model counting. The last family is taken from [Lee and Jiang, 2014], which encodes the *maximum probabilistic equivalence checking* (MPEC) of analyzing the maximum probability for a probabilistic circuit to produce erroneous outputs.

Table 1 shows the results, where benchmarks statistics, including the numbers of variables (#V), clauses (#C), outermost existentially quantified variables (#E_1), randomized quantified variables (#R), and innermost existentially quantified variables (#E_2), are reported. In the table, runtime information is reported in seconds. For erSSAT, the obtained lower bound (LB), the time (T_1) spent to first reach the obtained LB, and the entire runtime (T_2), are reported. For Dc, the exact probability (Pr) and runtime (T) are reported. As Dc is an exact SSAT solver, either it terminates and returns an answer, or it times out without producing any information. For Max, the lower bound (LB), confidence level (CL) of LB, and the rumtime (T) are reported. For c2d, the time (T) spent on compiling a formula into d-DNNF is reported. A "-" entry in T, T_1, or T_2 indicates that the time limit was reached. We note that, if T_2 of erSSAT is not a "-", the corresponding LB is the exact answer to the formula. Also the LB values obtained by erSSAT are of confidence level 1.

The results suggest that erSSAT was able to quickly derive tight lower bounds in families Toilet-A, Conformant, Sand-Castle, MaxSat, and MPEC. Specifically, for Toilet-A, erSSAT generally spent less time than Max did to derive the same lower bounds while Max timed out on four larger cases. For Conformant, erSSAT derived lower bounds at the order of 10^{-1} for all formulas, while Dc, Max, and c2d timed out on most of them. For Sand-Castle, erSSAT exactly solved three formulas, and derived lower bounds greater than 0.99 for the rest. In contrast, Dc and c2d outperformed erSSAT on these instances because they are specially designed for solving probabilistic conformant planning tasks. On the other

benchmark statistics							erSSAT			Dc		Max			c2d
family	formula	#V	#C	#E_1	#R	#E_2	LB	T_1	T_2	Pr	T	LB	CL	T	T
Toilet-A	10_01.3	106	10604	33	10	63	**1.95e-3**	0	27	**1.95e-3**	13	**1.95e-3**	1.00	36	3
	10_01.5	170	10902	55	10	105	**3.91e-3**	19	577	**3.91e-3**	208	**3.91e-3**	1.00	67	5
	10_01.7	234	11200	77	10	147	**7.81e-3**	179	-	-	-	**7.81e-3**	1.00	294	19
	10_05.2	170	11315	110	10	50	**3.13e-2**	565	-	-	-	-	-	-	
	10_05.3	250	12000	165	10	75	**1.56e-2**	0	-	-	-	-	-	-	244
	10_05.4	330	12685	220	10	100	**1.56e-2**	888	-	-	-	-	-	-	-
	10_10.2	290	12840	220	10	60	**1.00**	3	3	-	-	-	-	-	181
Conformant	blocks_enc_2_b4	3043	57130	1248	7	1788	**4.38e-1**	341	-	-	-	-	-	-	-
	cube_c7_ser—23	1479	15164	138	9	1332	**3.38e-1**	620	-	-	-	-	-	-	-
	cube_c7_ser–opt-24	1542	15510	144	9	1389	**3.44e-1**	679	-	-	-	-	-	-	-
	cube_c9_par—10	847	24106	60	10	777	2.90e-1	185	-	-	-	**2.92e-1**	1.00	802	-
	cube_c9_par–opt-11	928	24548	66	10	852	**2.89e-1**	192	-	-	-	-	-	-	-
	emptyroom_e3_ser–20	982	6286	80	6	896	**1.88e-1**	869	-	-	-	-	-	-	-
	ring_r4_ser–opt-11	373	5333	44	9	320	**4.96e-1**	506	-	-	-	4.53e-1	1.00	102	29
Sand-Castle	SC-11	101	201	22	55	24	**9.77e-1**	32	50	**9.77e-1**	0	-	-	-	0
	SC-12	110	219	24	60	26	**9.84e-1**	133	187	**9.84e-1**	0	-	-	-	0
	SC-13	119	237	26	65	28	**9.89e-1**	441	619	**9.89e-1**	0	-	-	-	0
	SC-14	128	255	28	70	30	**9.92e-1**	632	-	**9.92e-1**	1	-	-	-	0
	SC-15	137	273	30	75	32	9.93e-1	979	-	**9.94e-1**	1	-	-	-	1
	SC-16	146	291	32	80	34	9.94e-1	785	-	**9.96e-1**	3	-	-	-	0
	SC-17	155	309	34	85	36	9.94e-1	654	-	**9.97e-1**	6	-	-	-	1
MaxSat	keller4.clq	120	1212	43	15	62	**9.76e-1**	0	0	-	-	9.13e-1	0.82	5	1
QIF	backdoor-2x16-8	200	272	32	32	136	**5.96e-8**	0	-	-	-	**5.96e-8**	1.00	9	1
	backdoor-32-24	147	76	32	32	83	**1.00**	0	0	-	-	1.95e-3	0.82	601	0
	bin-search-16	1448	5825	16	16	1416	1.95e-3	106	-	-	-	**9.85e-1**	0.91	230	-
	CVe-2007-2875	784	1740	32	32	720	**1.00**	2	2	-	-	9.85e-1	0.82	13	342
	pwd-backdoor	400	609	64	64	272	0.00	-	-	-	-	**9.85e-1**	0.99	93	1
	reverse2	333	293	32	32	269	**2.98e-7**	271	-	-	-	-	-	-	2
	reverse	229	293	32	32	165	**5.96e-7**	839	-	-	-	-	-	-	2
PS	ConcreteActService	4836	17866	71	37	4728	0.00	-	-	-	-	**9.60e-1**	0.82	52	-
	IssueServiceImpl	3625	13028	77	29	3519	0.00	-	-	-	-	**9.06e-1**	0.82	34	-
	IterationService	4167	15264	70	34	4063	0.00	-	-	-	-	**9.70e-1**	0.82	47	-
	LoginService	5229	21566	92	27	5110	0.00	-	-	-	-	**9.45e-1**	0.82	56	-
	PhaseService	4167	15264	70	34	4063	0.00	-	-	-	-	**9.70e-1**	0.82	47	-
	ProcessBean	9880	41451	166	39	9675	0.00	-	-	-	-	**9.27e-1**	0.82	126	-
	UserServiceImpl	4019	14657	87	31	3901	0.00	-	-	-	-	**9.22e-1**	0.82	43	-
MPEC	c499(2.34e-1)	217	522	41	2	174	**2.34e-1**	0	0	**2.34e-1**	0	**2.34e-1**	1.00	0	2
	c880(2.34e-1)	451	1167	60	2	389	**1.25e-1**	0	-	-	-	**1.25e-1**	1.00	14	72
	c1355(3.30e-1)	771	2181	41	3	727	**3.30e-1**	0	-	-	-	**3.30e-1**	1.00	41	10
	c1908(2.34e-1)	270	705	33	2	235	**2.34e-1**	23	-	**2.34e-1**	91	1.25e-1	1.00	1	3
	c3540(1.25e-1)	321	807	50	2	269	**1.25e-1**	0	-	**1.25e-1**	92	**1.25e-1**	1.00	2	3
	c5315(7.37e-1)	918	2190	178	10	730	4.14e-1	154	-	-	-	**6.27e-1**	0.82	63	217
	c7552(4.87e-1)	648	1308	207	5	436	**2.34e-1**	0	-	-	-	2.18e-1	0.82	66	5
Maximum memory usage (GB)									2.2		38.6			0.2	4.2

Table 1: Results of Application Formulas

hand, Max failed in deriving lower bounds for all the formulas. For families QIF and PS, erSSAT did not perform as good as Max due to the fact that the formulas in these two families share the property that only very few assignments to the outermost existentially quantified variables can lead to large satisfying probabilities close to 1. It is not surprising as Max is particularly designed for such formulas.

In terms of memory usage, erSSAT is more efficient than Dc and c2d while Max used the least amount.

5.3 Analysis of Enhancement Techniques

To analyze the effect of the proposed enhancement techniques on computational efficiency, we tested our program under different configurations over random k-CNF formulas. Let the enabled enhancement techniques be indicated by letter m for minimal clause selection, s for clause subsumption, and p for partial assignment pruning. On average, erSSAT-{p} achieved 31% speedup over erSSAT; erSSAT-{m,p} achieved 11% speedup over erSSAT-{p}; erSSAT-{m,s,p} achieved 2% speedup over erSSAT-{m,p}.

For application formulas, the enhancement techniques in general do not produce consistent runtime improvements due to their computational overheads. However they helped in exact solving of more formulas within the timeout limit. The results suggest the benefit of the three enhancement techniques to the efficiency of the proposed algorithm.

6 Conclusions

We developed a new approach to solving E-MAJSAT formulas. In contrast to prior methods based on DPLL search or knowledge compilation, we proposed the clause containment learning technique, inspired by clause selection recently developed in QBF evaluation, and design a novel algorithm to solve E-MAJSAT efficiently. Under the framework of clause containment learning, three enhancement techniques were proposed to improve the computational efficiency. Experiment results show the benefit of our method. For future work, we intend to solve SSAT with general prefix structure.

Acknowledgements

This work was supported in part by the Ministry of Science and Technology of Taiwan under grants 104-2628-E-002-013-MY3, 105-2221-E-002-196-MY3, 105-2923-E-002-016-MY3, and 106-2912-E-002-002-MY3.

References

[Chakraborty *et al.*, 2016] S. Chakraborty, K. Meel, and M. Vardi. Algorithmic improvements in approximate counting for probabilistic inference: From linear to logarithmic SAT calls. In *Proc. IJCAI*, pages 3569–3576, 2016.

[Chavira and Darwiche, 2008] M. Chavira and A. Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

[Darwiche and Marquis, 2002] A. Darwiche and P. Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[Darwiche, 2001] A. Darwiche. Decomposable negation normal form. *Journal of the ACM*, 48(4):608–647, 2001.

[Darwiche, 2002] A. Darwiche. A compiler for deterministic, decomposable negation normal form. In *Proc. AAAI*, pages 627–634, 2002.

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

[Dechter, 1998] R. Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Learning in Graphical Models*, pages 75–104. Springer, 1998.

[Eén and Sörensson, 2003a] N. Eén and N. Sörensson. An extensible SAT-solver. In *Proc. SAT*, pages 502–518, 2003.

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

[Fremont *et al.*, 2017] D. Fremont, M. Rabe, and S. Seshia. Maximum Model Counting. In *AAAI*, pages 3885–3892, 2017.

[Giunchiglia *et al.*, 2005] E. Giunchiglia, M. Narizzano, L. Pulina, and A. Tacchella. Quantified Boolean formulas satisfiability library (QBFLIB), 2005. https://www.qbflib.org.

[Hnich *et al.*, 2011] B. Hnich, R. Rossi, A. Tarim, and S. Prestwich. A survey on CP-AI-OR hybrids for decision making under uncertainty. In *Hybrid Optimization*, pages 227–270. Springer, 2011.

[Huang, 2006] J. Huang. Combining knowledge compilation and search for conformant probabilistic planning. In *Proc. ICAPS*, pages 253–262, 2006.

[Janota and Marques-Silva, 2015] M. Janota and J. Marques-Silva. Solving QBF by clause selection. In *Proc. IJCAI*, pages 325–331, 2015.

[Lauria, 2012] M. Lauria. CNFgen: Combinatorial benchmarks for SAT solvers, 2012. https://massimolauria.github.io/cnfgen/.

[Lee and Jiang, 2014] N.-Z. Lee and J.-H. Jiang. Towards formal evaluation and verification of probabilistic design. In *Proc. ICCAD*, pages 340–347, 2014.

[Lee *et al.*, 2017] N.-Z. Lee, Y.-S. Wang, and J.-H. Jiang. Solving stochastic Boolean satisfiability under random-exist quantification. In *Proc. IJCAI*, pages 688–694, 2017.

[Littman *et al.*, 1998] M. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9(1):1–36, 1998.

[Littman *et al.*, 2001] M. Littman, S. Majercik, and T. Pitassi. Stochastic Boolean satisfiability. *Journal of Automated Reasoning*, 27(3):251–296, 2001.

[Majercik and Boots, 2005] S. Majercik and B. Boots. DC-SSAT: A divide-and-conquer approach to solving stochastic satisfiability problems efficiently. In *Proc. AAAI*, page 416, 2005.

[Majercik and Littman, 1998] S. Majercik and M. Littman. MAXPLAN: A new approach to probabilistic planning. In *Proc. AIPS*, pages 86–93, 1998.

[Majercik, 2009] S. Majercik. Stochastic Boolean satisfiability. *Handbook of Satisfiability*, 185:887–925, 2009.

[Mishchenko, 2005] A. Mishchenko. ABC: A system for sequential synthesis and verification, 2005. https://github.com/berkeley-abc/abc.

[Papadimitriou, 1985] C. Papadimitriou. Games against nature. *Journal of Computer and System Sciences*, 31(2):288–301, 1985.

[Pipatsrisawat and Darwiche, 2009] K. Pipatsrisawat and A. Darwiche. A new d-DNNF-based bound computation algorithm for functional E-MAJSAT. In *Proc. IJCAI*, pages 590–595, 2009.

[Rabe and Tentrup, 2015] M. Rabe and L. Tentrup. CAQE: A certifying QBF solver. In *Proc. FMCAD*, pages 136–143, 2015.

[Sang *et al.*, 2004] T. Sang, F. Bacchus, P. Beame, H. Kautz, and T. Pitassi. Combining component caching and clause learning for effective model counting. In *Proc. SAT*, pages 20–28, 2004.

[Sang *et al.*, 2005] T. Sang, P. Beame, and H. Kautz. Performing Bayesian inference by weighted model counting. In *Proc. AAAI*, pages 475–481, 2005.

[Somenzi, 1998] F. Somenzi. CUDD: CU decision diagram package release 2.3.0, 1998.