BAXMC: a CEGAR approach to Max#SAT

Thomas Vigouroux*, Cristian Ene*, David Monniaux*, Laurent Mounier*, Marie-Laure Potet*
*Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Grenoble, France
Firstname.Lastname@univ-grenoble-alpes.fr

***Abstract*—Max#SAT is an important problem with multiple applications in security and program synthesis that is proven hard to solve. It is defined as: given a parameterized quantifier-free propositional formula compute parameters such that the number of models of the formula is maximal. As an extension, the formula can include an existential prefix.**

We propose a CEGAR-based algorithm and refinements thereof, based on either exact or approximate model counting, and prove its correctness in both cases. Our experiments show that this algorithm has much better effective complexity than the state of the art.

I. INTRODUCTION

#SAT is the problem of counting the solutions of a quantifier-free propositional formula, the counting version of the SAT problem. Max#SAT is the problem of optimizing, according to some propositional variables, the number of solutions according to the others. We generalize this problem to allow an existential prefix in the formula.

This problem has many practical applications in diverse areas of computer science such as *quantitative program analysis* and *program synthesis* [1]. Most approaches for quantitative information flow analysis use approximations, with fast yet imprecise solutions. Adaptive attacker synthesis [2] would also benefit from advances in Max#SAT efficiency, mainly by being able to avoid the use of imprecise heuristics.

Unfortunately, Max#SAT has high complexity [3], [4], and practical solving methods remain costly. At the time of writing, only one solver is publicly available off-the-shelf [1].

Earlier work on the Max#SAT problem proposed two approaches. The first is a probabilistic solving method [1], which unfortunately degrades to exhaustive search when seeking precise answers to the problem. The second approach [5] solves the problem exactly, but scales poorly.

We present in this paper a new approach to Max#SAT, leveraging ideas from CEGAR solvers, and show its effectiveness on various benchmarks used in previous publications on the subject. We also present improvements of our algorithm based on previous work about symmetry breaking in SAT solvers [6].

Our contributions are the following:

- An *effective algorithm* to compute maximal solutions for the projected model counting problem (Sections III and IV). This algorithm relies either on an *exact* projected model counter as a subprocedure, or on an *approximated* one, which should be the case most times in practice for scalability reasons. A complete correctness proof of this algorithm is given for both cases.
- The *extension* of our algorithm with SAT *symmetry breaking* techniques (Section V) and *heuristics* (Section VI), to further improve its efficiency.
- The *implementation* of this algorithm in the tool BAXMC [7], together with a set of *experimental results* (Section VII), showing the accuracy and performances of our Max#SAT algorithm on various benchmarks, with respect to the only other available tool.

This work was partially supported by the French ANR project TAVA (ANR-20-CE25-0009) and by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01) funded by the French program Investissements d'avenir.

II. PRELIMINARIES

We set our problem in standard Boolean logic. Throughout the paper, Greek letters (ϕ, ψ, ...) denote Boolean formulas, uppercase calligraphic Latin letters ($\mathcal{V}$, $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$, ...) denote sets of variables, simple uppercase Latin letters (V, X, Y, Z, ...) denote variables, lowercase variants of these letters (x, y, z, ...) denote valuations for these sets of variables.

Let $\mathbb{B} = \{true, false\}$. A *literal* is a variable or its negation and the set of literals derived from a set of variables $\mathcal{V}$ is denoted by $\overline{\mathcal{V}} = \mathcal{V} \cup \neg\mathcal{V}$. Let $\phi(\mathcal{V})$ be a Boolean formula over $\mathcal{V}$ a set of variables. A valuation $v : \mathcal{V} \to \mathbb{B}$ is a *model* of ϕ if ϕ evaluates to *true* over v; this is denoted by $v \models \phi$.

We say that a formula ϕ is *satisfiable* if there exists v such that $v \models \phi$. Otherwise, ϕ is deemed *unsatisfiable*. Determining whether a formula is satisfiable or unsatisfiable is called the *satisfiability* problem, also known as SAT.

The *restriction* of a valuation $v : \mathcal{V} \to \mathbb{B}$ to $\mathcal{E} \subseteq \mathcal{V}$ is denoted by $v|_{\mathcal{E}}$. We say that two valuations v_1 and v_2 *agree* on $\mathcal{E}$, denoted by $v_1 \sim_{\mathcal{E}} v_2$, if their restrictions to $\mathcal{E}$ are equal.

A. Base definitions

Definition II.1 (Equivalence class)**.** Given a valuation v and a set $\mathcal{E}$, we call *equivalence class of v over $\mathcal{E}$* the set of valuations that agree with v over $\mathcal{E}$, that is:

$$[v]_{\mathcal{E}} = \{v' \mid v' \sim_{\mathcal{E}} v\}$$

We call $v|_{\mathcal{E}}$ *partial* models, and v *complete* models. The elements of $[v]_{\mathcal{E}}$ are called the *extensions* or $v|_{\mathcal{E}}$.

Definition II.2. Given propositional formula $\phi(\mathcal{V})$ and $\mathcal{E} \subseteq \mathcal{V}$, $\mathcal{M}_{\mathcal{E}}(\phi) = \{v|_{\mathcal{E}} \mid v \models \phi\}$ denotes the *set of models projected over $\mathcal{E}$*.

Remark II.1. We omit the set $\mathcal{E}$ when it contains all the variables of ϕ. That is $\mathcal{M}(\phi) = \mathcal{M}_{\mathcal{V}}(\phi)$.

https://doi.org/10.34727/2022/isbn.978-3-85448-053-2_23

Definition II.3. Given a valuation v, we define the *update* of the variable $V \in \mathcal{V}$ to b as:

$$v[V \to b](X) = \begin{cases} v(X) & \text{if } X \neq V \\ b & \text{otherwise} \end{cases}$$

Definition II.4. Given two formulas $\phi(\mathcal{V})$ and $\psi(\mathcal{V})$, we say that ϕ *entails* ψ (denoted by $\phi \models \psi$) if $\mathcal{M}(\phi) \subseteq \mathcal{M}(\psi)$.

B. Domain-specific definitions

In the remaining of the paper we consider a *partition* of $\mathcal{V}$ over three sets $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, respectively called *witness*, *counting* and *intermediate* variables. Given a Boolean formula $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, we define Max#SAT as an optimization problem stated as follows: find $x_m \in \mathcal{M}_{\mathcal{X}}(\phi)$ such that the projected model counting over $\mathcal{Y}$ of the formula (which will be defined later) is maximal.

Definition II.5 (Induced set)**.** Given a formula $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ and $x \in \mathcal{M}_{\mathcal{X}}(\phi)$, the set of models over $\mathcal{Y}$ *induced* by x is:

$$I_\phi(x) = \{y \in \mathcal{M}_{\mathcal{Y}}(\phi) \mid \exists z \,.\, (x, y, z) \models \phi\}$$

We extend this definition to partial witnesses as follows: $I_\phi(x|_{\mathcal{E}}) = \bigcup_{x' \in [x]_{\mathcal{E}}} I_\phi(x')$.

Definition II.6 (Model counting)**.** Given a formula $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, the *count* of a witness x is defined by the size of the set it induces $|\phi(x, \mathcal{Y}, \mathcal{Z})| = |I_\phi(x)|$.

We extend this definition to partial witnesses as follows $|\phi(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z})| = |I_\phi(x|_{\mathcal{E}})|$.

Definition II.7 (Max#SAT)**.** Given formula $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, we can state the Max#SAT problem more formally as finding $x_m \in \mathcal{M}_{\mathcal{X}}(\phi)$ such that:

$$|\phi(x_m, \mathcal{Y}, \mathcal{Z})| = \max_{x \in \mathcal{M}_{\mathcal{X}}(\phi)} |\phi(x, \mathcal{Y}, \mathcal{Z})|$$

Property II.1. Given a formula $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, the count of a partial witness is an upper-bound of the count of its extensions:

$$\forall x' \in [x]_{\mathcal{E}}, |\phi(x', \mathcal{Y}, \mathcal{Z})| \leq |\phi(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z})|$$

Proof. This follows directly from Definition II.5 on induced sets. □

Property II.2 (Monotony of model counting)**.** Given a propositional formula $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{X}$, and $x \in \mathcal{M}_{\mathcal{X}}(\phi)$, the count of partial solutions is monotonous:

$$|\phi(x|_{\mathcal{B}}, \mathcal{Y}, \mathcal{Z})| \leq |\phi(x|_{\mathcal{A}}, \mathcal{Y}, \mathcal{Z})|$$

Proof. First, following Definition II.1 we have:

$$[x]_{\mathcal{B}} \subseteq [x]_{\mathcal{A}}$$

Hence, following Definition II.5:

$$I_\phi(x|_{\mathcal{B}}) \subseteq I_\phi(x|_{\mathcal{A}})$$

□

Property II.3. Given a Boolean formula $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ such that $x \in \mathcal{M}_{\mathcal{X}}(\phi)$, $\mathcal{E} \subseteq \mathcal{X}$ and $X_i \in \mathcal{X}$, we have:

$$|\phi(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z})| \leq \left|\phi\left(x|_{\mathcal{E}-\{X_i\}}, \mathcal{Y}, \mathcal{Z}\right)\right| \leq |\phi(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z})| + |\phi(x|_{\mathcal{E}}[X_i \to \neg x(X_i)], \mathcal{Y}, \mathcal{Z})|$$

Proof. The first inequality is a direct consequence of Property II.2. The last inequality follows from Definition II.5. □

Property II.4. For a given $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ and $\phi'(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ such that $\phi \models \phi'$, a witness x, and $\mathcal{E} \subseteq \mathcal{X}$ we have:

$$|\phi(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z})| \leq |\phi'(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z})|$$

Proof. For any $y \in I_\phi(x|_{\mathcal{E}})$, as $y \models \phi$, and $\phi \models \phi'$, we get $y \models \phi'$ and hence $I_\phi(x|_{\mathcal{E}}) \subseteq I_{\phi'}(x|_{\mathcal{E}})$. □

Property II.5. Given $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, $\psi(\mathcal{X})$ and $x \models \psi$, we have:

$$|\phi(x, \mathcal{Y}, \mathcal{Z})| = |(\phi \wedge \psi)(x, \mathcal{Y}, \mathcal{Z})|$$

Proof. Since $x \models \psi$ and ψ does not depend on $\mathcal{Y}$ and $\mathcal{Z}$, $(x, y, z) \models \phi$ if and only if $(x, y, z) \models \phi \wedge \psi$. □

III. Solving Max#SAT

This section presents the main algorithm we propose to solve the Max#SAT problem.

A. The main algorithm

Algorithm 1 takes as input a formula $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ and computes a pair (x_m, n_m) such that x_m is a solution to Max#SAT for ϕ with model counting n_m. Together with the formula, the algorithm takes multiple precision parameters:

- (ϵ_i) that are called *tolerance* parameters [8];
- (δ_i) that are called *confidence* parameters [8];
- κ that is called the *persistence* parameter.

Further explanations about these parameters will be given later.

Roughly speaking, this algorithm consists in iterating over possible *witnesses* x of ϕ. If the model count for x is less than the current best solution, it *blocks generalizations* of x such that all extensions of these generalizations are *worse* than the current best solution (Lines 14 and 16), hence removing a chunk of the search space at each iteration. Otherwise, it saves the candidate, which is then the new maximum, and blocks it (Lines 10 and 16), removing only one candidate from the search space.

We use two kinds of oracles in this algorithm. At Line 7 we call a SAT solver. Calls to an existing #SAT oracle (Lines 5, 8 and 17) can be performed using either an exact or an approximate model counter. In the latter case the precision parameters taken as input of the algorithm are used to configure the oracle, and influence the correctness of the returned value (in the former case, simply assume that they are all 0).

Definition III.1. Given $\phi(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, $x \in \mathcal{M}_{\mathcal{X}}(\phi)$ we say that $\mathcal{E} \subseteq \mathcal{X}$ is n-bounding if $|\phi(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z})| \leq n$.

The Generalize function used in Algorithm 1 at Line 14 is proved to return n_m-bounding sets in both the exact (Theorem IV.1) and the approximate case (with probability $1 - \delta$,

Algorithm 1 Pseudocode for the BAXMC algorithm

function $\text{BAXMC}_{\epsilon_0,\epsilon_1,\delta_0,\delta_1,\delta_2,\kappa}(\phi(\mathcal{X},\mathcal{Y},\mathcal{Z}))$
  $\phi_s \leftarrow \phi$
  $x_m \leftarrow \top$
  $n_m \leftarrow 0$
  $N \leftarrow \text{MC}_{\epsilon_0,\delta_0}\left(\phi\left(\emptyset,\mathcal{Y},\mathcal{Z}\right)\right)$
  while $n_m < \frac{N}{1+\kappa}$ **do**
    $x \xleftarrow{\$} \mathcal{M}_X(\phi_s)$ ▷ Pick a new candidate
    $c \leftarrow \text{MC}_{\epsilon_1,\delta_1}\left(\phi_s\left(x,\mathcal{Y},\mathcal{Z}\right)\right)$
    if $c > n_m$ **then** ▷ New maximum
      $x_m \leftarrow x$
      $n_m \leftarrow c$
      $\mathcal{E} \leftarrow \mathcal{X}$
    else ▷ Find generalization
      $\mathcal{E} \leftarrow \text{GENERALIZE}_{\delta_2}(x, \phi_s, n_m)$
    end if
    $\phi_s \leftarrow \phi_s \wedge \neg\left(x|_{\mathcal{E}}\right)$ ▷ Block
    $N \leftarrow \text{MC}_{\epsilon_0,\delta_0}\left(\phi_s\left(\emptyset,\mathcal{Y},\mathcal{Z}\right)\right)$
  end while
  return x_m, n_m
end function

Theorem IV.2). The GENERALIZE function is called Algorithm 2 in this paper and it will be presented in Section IV.

B. Termination and correctness with an exact #SAT oracle

In this subsection, each i-indexed variable of the algorithm denotes its value at the end of the i-th iteration of the main loop. In the exact version of the algorithm, all precision parameters are assumed to be equal to 0 and all calls to $\text{MC}_{0,0}\left(\phi\left(x,\mathcal{Y},\mathcal{Z}\right)\right)$ return $\left|\phi\left(x,\mathcal{Y},\mathcal{Z}\right)\right|$.

Theorem III.1 (Termination with an exact #SAT oracle)**.** *Algorithm 1 always terminates.*

Proof. By construction of $\left(\phi_{s_i}\right)_i$ we have:

$$\forall i > 0, 0 \leq \left|\mathcal{M}_{\mathcal{X}}\left(\phi_{s_{i+1}}\right)\right| < \left|\mathcal{M}_{\mathcal{X}}\left(\phi_{s_i}\right)\right|$$

The sequence $\left(n_{mi}\right)_i$ is obviously increasing. From Property II.4, the sequence $\left(N_i\right)_i$ is decreasing and hence $\left(N_i - n_{mi}\right)_i$ is decreasing.

Putting all this together, $\left(\left|\mathcal{M}_{\mathcal{X}}\left(\phi_{s_i}\right)\right| + \left(N_i - n_{mi}\right)\right)_i$ is strictly decreasing.

On can easily see that whenever $\left|\mathcal{M}_{\mathcal{X}}\left(\phi_{s_i}\right)\right| = 0$ it follows that $N_i = 0$ and $N_i - n_{mi} \leq 0$. Hence in all cases, after some iteration k, $N_k - n_{mk} \leq 0$ and the termination follows. □

Remark III.1. The worst case complexity of Algorithm 1 is reached when it iterates over all the witnesses of the formula.

Let k be the number of iterations performed when Algorithm 1 terminates, then we have $n_{mk} \geq N_k$.

Lemma III.1. *At every iteration i of Algorithm 1, we have:*

$$\mathcal{M}_{\mathcal{X}}\left(\phi_{s_i}\right) = \mathcal{M}_{\mathcal{X}}\left(\phi\right) - \bigcup_{j<i} \left[x_i\right]_{\mathcal{E}_i}$$

Proof. This follows by construction of ϕ_{s_i}. □

Lemma III.2. *At every iteration i of Algorithm 1, and assuming* GENERALIZE(x, ϕ, n) *returns n-bounding generalizations of x (as defined in Definition III.1) we have:*

$$\forall x' \in \bigcup_{j\leq i} \left[x_j\right]_{\mathcal{E}_j}, \left|\phi\left(x',\mathcal{Y},\mathcal{Z}\right)\right| \leq n_{mi}$$

Proof. Let $j \leq i$, and let $x \in \left[x_j\right]_{\mathcal{E}_j}$, following Definition III.1, we have $\left|\phi_{s_j}\left(x,\mathcal{Y},\mathcal{Z}\right)\right| \leq n_{mj}$.

Then by construction of ϕ_{si} and Property II.5 we have $\left|\phi\left(x,\mathcal{Y},\mathcal{Z}\right)\right| \leq n_{mj}$ which, as $\left(n_{mi}\right)_i$ is increasing, proves the lemma. □

Theorem III.2 (Correctness with an exact #SAT oracle)**.** *Algorithm 1 is correct, i.e., the returned tuple (x_m, n_m) satisfies the following relation:*

$$n_m = \left|\phi\left(x_m,\mathcal{Y},\mathcal{Z}\right)\right| = \max_{x\in\mathcal{M}_{\mathcal{X}}(\phi)} \left|\phi\left(x,\mathcal{Y},\mathcal{Z}\right)\right|$$

Proof. Following Property II.1 and since $n_{mk} \geq N_k$ we have:

$$\forall x \in \mathcal{M}_{\mathcal{X}}\left(\phi_{s_k}\right) \ . \ \left|\phi\left(x,\mathcal{Y},\mathcal{Z}\right)\right| \leq N_k \leq n_{mk}$$

Then instantiating Lemma III.2 at iteration k we have:

$$\forall x \in \bigcup_{i\leq k} \left[x_i\right]_{\mathcal{E}_i}, \left|\phi(x,Y,Z)\right| \leq n_{mk}$$

Following Lemma III.1, at iteration k we have $\mathcal{M}_{\mathcal{X}}\left(\phi\right) = \mathcal{M}_{\mathcal{X}}\left(\phi_{s_k}\right) \cup \bigcup_{i\leq k} \left[x_i\right]_{\mathcal{E}_i}$, and the result follows. □

C. Correctness with a probabilistic #SAT oracle

Since the termination can be proven in the same way as in the exact case, we only prove the correctness.

Let us first recall the expected guarantees provided by an approximate model counter [8], where the ϵ parameter characterizes the precision of the result and the δ parameter determines its associated confidence.

Property III.1 (Correctness of the Model Counting)**.** The count $\text{MC}_{\epsilon,\delta}\left(\phi\left(x,\mathcal{Y},\mathcal{Z}\right)\right)$ returned by an approximate model counter satisfies the following:

$$\mathbb{P}\left[\frac{1}{1+\epsilon} \leq \frac{\text{MC}_{\epsilon,\delta}\left(\phi\left(x,\mathcal{Y},\mathcal{Z}\right)\right)}{\left|\phi(x,\mathcal{Y},\mathcal{Z})\right|} \leq 1+\epsilon\right] \geq 1-\delta$$

These guarantees extend to partial witnesses naturally, i.e., queries of the form $\text{MC}_{\epsilon,\delta}\left(\phi\left(x|_{\mathcal{E}},\mathcal{Y},\mathcal{Z}\right)\right)$.

The next theorem proves the correctness of Algorithm 1 in the approximate case and gives the associated *tight* bounds.

Theorem III.3. *Let (x_m, n_m) be the result returned by the call* $\text{BAXMC}_{\epsilon_0,\epsilon_1,\delta_0,\delta_1,\delta_2,\kappa}(\phi(\mathcal{X},\mathcal{Y},\mathcal{Z}))$*, and let*

$$M = \max_{x\in\mathcal{M}_{\mathcal{X}}(\phi)} \left|\phi\left(x,\mathcal{Y},\mathcal{Z}\right)\right|$$

If $\delta_1 \leq \frac{\delta_2}{|\mathcal{X}|+1}$ then:

$$\mathbb{P}\left[\frac{1}{1+\epsilon_1} \leq \frac{n_m}{\left|\phi\left(x_m,\mathcal{Y},\mathcal{Z}\right)\right|} \leq 1+\epsilon_1\right] \geq 1-\delta_1$$

and

$$\mathbb{P}\left[|\phi\left(x_m, \mathcal{Y}, \mathcal{Z}\right)| \geq \frac{M}{(1+\epsilon_0)*(1+\epsilon_1)*(1+\kappa)}\right] \geq (1-\delta_1)*\min(1-\delta_2, 1-\delta_0)$$

Proof. Let ϕ_S be the final value of the variable ϕ_s after the last iteration of the **while** loop. We have the following guarantees from the approximate model counter (Property III.1):

$$\mathbb{P}\left[\frac{1}{1+\epsilon_1} \leq \frac{n_m}{|\phi\left(x_m, \mathcal{Y}, \mathcal{Z}\right)|} \leq 1+\epsilon_1\right] \geq 1-\delta_1 \quad (1)$$

$$\mathbb{P}\left[\frac{1}{1+\epsilon_0} \leq \frac{N}{|\mathcal{M}_{\mathcal{Y}}\left(\phi_S\right)|} \leq 1+\epsilon_0\right] \geq 1-\delta_0 \quad (2)$$

From Theorem IV.2 regarding the GENERALIZE function (which will be proved in the next section), we also have that for any $x \in \mathcal{M}_{\mathcal{X}}\left(\phi \wedge \neg\phi_S\right)$ it holds (assuming that $\delta_1 \leq \frac{\delta_2}{|\mathcal{X}|+1}$):

$$\mathbb{P}\left[|\phi\left(x, \mathcal{Y}, \mathcal{Z}\right)| \leq n_m\right] \geq 1-\delta_2. \quad (3)$$

After the last iteration of the **while** loop we have that $n_m * (1+\kappa) \geq N$. Using this and Equation (2) and Property II.5 we get that for any $x \in \mathcal{M}_{\mathcal{X}}\left(\phi_S\right)$ it holds

$$\begin{aligned}\mathbb{P}\left[|\phi\left(x, \mathcal{Y}, \mathcal{Z}\right)| \leq n_m*(1+\kappa)*(1+\epsilon_0)\right] &\geq \\ \mathbb{P}\left[|\phi\left(x, \mathcal{Y}, \mathcal{Z}\right)| \leq N*(1+\epsilon_0)\right] &= \\ \mathbb{P}\left[|\phi_S\left(x, \mathcal{Y}, \mathcal{Z}\right)| \leq N*(1+\epsilon_0)\right] &\geq \\ \mathbb{P}\left[|\mathcal{M}_{\mathcal{Y}}\left(\phi_S\right)| \leq N*(1+\epsilon_0)\right] &\geq 1-\delta_0.\end{aligned}$$

From Equation (3), for any $x \in \mathcal{M}_{\mathcal{X}}\left(\phi \wedge \neg\phi_S\right)$ it holds

$$\begin{aligned}\mathbb{P}\left[|\phi\left(x, \mathcal{Y}, \mathcal{Z}\right)| \leq n_m*(1+\kappa)*(1+\epsilon_0)\right] &\geq \\ \mathbb{P}\left[|\phi\left(x, \mathcal{Y}, \mathcal{Z}\right)| \leq n_m\right] &\geq 1-\delta_2.\end{aligned}$$

Hence, for any $x \in \mathcal{M}_{\mathcal{X}}\left(\phi\right)$ it holds

$$\mathbb{P}\left[|\phi\left(x, \mathcal{Y}, \mathcal{Z}\right)| \leq n_m*(1+\kappa)*(1+\epsilon_0)\right] \geq \min(1-\delta_2, 1-\delta_0)$$

and hence

$$\mathbb{P}\left[\frac{n_m}{1+\epsilon_1} \geq \frac{M}{(1+\kappa)*(1+\epsilon_0)*(1+\epsilon_1)}\right] \geq \min(1-\delta_2, 1-\delta_0). \quad (4)$$

Combining this with the Equation (1), we obtain

$$\mathbb{P}\left[|\phi\left(x_m, \mathcal{Y}, \mathcal{Z}\right)| \geq \frac{M}{(1+\kappa)*(1+\epsilon_0)*(1+\epsilon_1)}\right] \geq (1-\delta_1)*\min(1-\delta_2, 1-\delta_0).$$ □

The following corollary instantiates Theorem III.3 in order to get the standard form (as in Property III.1).

Corollary III.1. *For any $0 < \epsilon, \delta < 1$, if in the call of the* BAXMC *function, we take as parameters $\epsilon_0 = \epsilon_1 = \kappa = \sqrt[3]{1+\epsilon} - 1$, $\delta_0 = \delta_2 = \frac{\delta}{2}$ and $\delta_1 = \frac{\delta}{2*(|\mathcal{X}|+1)}$, then the result (x_m, n_m) satisfies the following inequalities:*

$$\mathbb{P}\left[\frac{1}{1+\epsilon} \leq \frac{n_m}{|\phi\left(x_m, \mathcal{Y}, \mathcal{Z}\right)|} \leq 1+\epsilon\right] \geq 1-\delta$$

$$\mathbb{P}\left[|\phi\left(x_m, \mathcal{Y}, \mathcal{Z}\right)| \geq \frac{M}{1+\epsilon}\right] \geq 1-\delta$$

where

$$M = \max_{x \in \mathcal{M}_{\mathcal{X}}(\phi)} |\phi\left(x, \mathcal{Y}, \mathcal{Z}\right)|$$

Proof. It is easy to check that $\epsilon_1 = \sqrt[3]{1+\epsilon} - 1 \leq \epsilon$, $\delta_1 = \frac{\delta}{2*(|\mathcal{X}|+1)} < \delta_2 = \frac{\delta}{2} < \delta$, $(1+\epsilon_0)^3 = 1+\epsilon$ and $(1-\delta_1)*(1-\delta_0) = 1-\delta_0-\delta_1+\delta_0*\delta_1 > 1-2*\delta_0 = 1-\delta$. □

IV. GENERALIZATION ALGORITHM

Algorithm 2 generalizes a single model x with insufficiently high count to a set of models with insufficiently high count. This is much the same that a CDCL loop blocks not only one assignment, but a whole set of assignments.

As shown in Property II.2, generalizing a witness is an instance of the MSMP problem (*Minimal Set subject to a Monotone Predicate*), which can be solved using generic algorithms such as QUICKXPLAIN [9]. Although in theory this should lead to a better algorithm, in practice we observed larger numbers of calls to the #SAT oracle, an issue already identified in other contexts [10].

Algorithm 2 is thus a specific solver of the MSMP problem in our setting, relying on a *linear sweep* over the variables that are part of the valuation.

For efficiency reasons, the steps mentioned in Algorithm 2 are in a precise order. The reason behind this is:

1) The first step relies on a consequence of Property II.3, allowing to relax variables with simple calls to a sat solver.
2) The log-based generalization is a heuristic allowing to do *big steps* in the generalization process by relaxing multiple variables at each loop turn.
3) The *linear sweep* pass generalizes x in such a way that the returned set is minimal, i.e. that none of the further generalizations of the returned value satisfies Definition III.1.

The returned $\mathcal{E}$ is guaranteed only to be a *local minimum* and it may not be the *smallest* set such that Definition III.1 holds because of the order in which we consider variables of $\mathcal{X}$ in Algorithm 2.

A. Correctness and complexity with an exact #SAT oracle

Property IV.1. If $\phi(x|_{\mathcal{E}}[X_i \rightarrow \neg x(X_i)], \mathcal{Y}, \mathcal{Z})$ is UNSAT then:

$$|\phi\left(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z}\right)| = \left|\phi\left(x|_{\mathcal{E}-\{X_i\}}, \mathcal{Y}, \mathcal{Z}\right)\right|$$

Proof. This follows directly from Property II.3. □

Let us prove the correctness of Algorithm 2 in the context of an exact #SAT oracle. This will finish the correctness proof started in Section III-B.

Algorithm 2 Pseudocode for the generalization algorithm

1: **function** GENERALIZE$_\delta(x,\ \phi(\mathcal{X},\mathcal{Y},\mathcal{Z}),\ n_m)$
2: $\quad\mathcal{E} \leftarrow \mathcal{X}$
3: $\quad$**for all** $X_i \in \mathcal{X}$ **do** ▷ Redundancy elimination
4: $\quad\quad$**if** $\phi\left(x[X_i \rightarrow \neg x(X_i)], \mathcal{Y}, \mathcal{Z}\right)$ UNSAT **then**
5: $\quad\quad\quad\mathcal{E} \leftarrow \mathcal{E} - \{X_i\}$
6: $\quad\quad$**end if**
7: $\quad$**end for**
8: $\quad k \leftarrow \log n_m - \log \mathrm{MC}_{\epsilon,\delta_1}\left(\phi\left(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z}\right)\right)$
9: $\quad$**while** $k > 0 \wedge |\mathcal{E}| > 0$ **do** ▷ Log-elimination
10: $\quad\quad\mathcal{A}_k \overset{\$}{\leftarrow} \{\mathcal{V} \subseteq \mathcal{E} \mid |\mathcal{V}| = k\}$
11: $\quad\quad c \leftarrow \mathrm{MC}_{\epsilon,\delta_1}\left(\phi\left(x|_{\mathcal{E}-\mathcal{A}_k}, \mathcal{Y}, \mathcal{Z}\right)\right)$
12: $\quad\quad$**if** $c \leq \frac{n_m}{1+\epsilon}$ **then**
13: $\quad\quad\quad\mathcal{E} \leftarrow \mathcal{E} - \mathcal{A}_k$
14: $\quad\quad\quad k \leftarrow \log n_m - \log c$
15: $\quad\quad$**else**
16: $\quad\quad\quad k \leftarrow k - 1$
17: $\quad\quad$**end if**
18: $\quad$**end while**
19: $\quad$**for all** $X_i \in \mathcal{X} - \mathcal{E}$ **do** ▷ Refinement
20: $\quad\quad$**if** $\mathrm{MC}_{\epsilon,\delta_1}\left(\phi\left(x|_{\mathcal{E}-\{X_i\}}, \mathcal{Y}, \mathcal{Z}\right)\right) \leq \frac{n_m}{1+\epsilon}$ **then**
21: $\quad\quad\quad\mathcal{E} \leftarrow \mathcal{E} - \{X_i\}$
22: $\quad\quad$**end if**
23: $\quad$**end for**
24: **end function**

Theorem IV.1. *Algorithm 2 terminates and is correct: the returned set $\mathcal{E}$ satisfies Definition III.1, i.e., $|\phi\left(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z}\right)| \leq n$.*

Proof. In the **while** loop at Line 9 we can see that, at each iteration, either $|\mathcal{E}|$ or k decreases, thus ensuring the termination of the algorithm.

During any update of the temporary value $\mathcal{E}$ (Lines 5, 13 and 21), we ensure that the new value of $\mathcal{E}$ satisfies Definition III.1:

1) At Line 5, Property IV.1 keeps the model counting stable.
2) At Lines 13 and 21, the update is guarded by the explicit check of the property (in the **if** statement Lines 12 and 20).

Hence the correctness follows. □

B. Bounds with an approximate #SAT oracle

Theorem IV.2. *Let $\mathcal{E} \subseteq \mathcal{X}$ be the set returned by the call* GENERALIZE$_\delta(x, \phi(\mathcal{X},\mathcal{Y},\mathcal{Z}), n)$*, and assume that*

$$\mathbb{P}\left[|\phi\left(x, \mathcal{Y}, \mathcal{Z}\right)| \leq n\right] \geq 1 - \frac{\delta}{|\mathcal{X}|+1}$$

Then:

$$\mathbb{P}\left[|\phi\left(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z}\right)| \leq n\right] \geq 1 - \delta$$

Proof. Using Property IV.1, the variable $\mathcal{E}$ after the first loop within Algorithm 2 satisfies $|\phi\left(x, \mathcal{Y}, \mathcal{Z}\right)| = |\phi\left(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z}\right)|$.

We denote by $C_{x|_{\mathcal{V}}}$ the value returned by the call $\mathrm{MC}_{\epsilon,\delta_1}\left(\phi\left(x|_{\mathcal{V}}, \mathcal{Y}, \mathcal{Z}\right)\right)$. Since each time we update $\mathcal{E}$ to a set $\mathcal{V}$ we ensure $C_{x|_{\mathcal{V}}} \leq \frac{n}{1+\epsilon}$, we have the following probability:

$$\mathbb{P}\left[|\phi\left(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z}\right)| \leq n\right] \geq 1 - \delta_1$$

Let $\mathcal{E}_l$ denote the value obtained after l updates of variable $\mathcal{E}$ during LOG-ELIMINATION and REFINEMENT steps within Algorithm 2 and let us denote by P_l the probability that the set $\mathcal{E}_l$ is approximately n-bounding.

Using that we update $\mathcal{E}$ to the value $\mathcal{E}_l$ only if $C_{x|_{\mathcal{E}_l}} * (1+\epsilon) \leq n$, we have the following recursive relation:

$$\begin{aligned} P_l &= \mathbb{P}\left[|\phi\left(x|_{\mathcal{E}_l}, \mathcal{Y}, \mathcal{Z}\right)| \leq n\right] * P_{l-1} \\ &\geq (1-\delta_1) * P_{l-1} \geq (1-\delta_1)^l * P_0 \\ &\geq (1-\delta_1)^l * \left(1 - \frac{\delta}{|\mathcal{X}|+1}\right) \end{aligned}$$

Thus, as $l \leq |\mathcal{X}|$, if we take $\delta_1 = \frac{\delta}{|\mathcal{X}|+1}$ and we call the #SAT oracle with parameters $(\epsilon, \frac{\delta}{|\mathcal{X}|+1})$ we get:

$$\mathbb{P}\left[|\phi\left(x|_{\mathcal{E}}, \mathcal{Y}, \mathcal{Z}\right)| \leq n\right] \geq (1-\delta_1)^{l+1} \geq 1-(l+1)*\delta_1 \geq 1-\delta$$

□

Remark IV.1. The bound with respect to the number of updates is tight. The worst case is reached when the only valid subset of $\mathcal{X}$ is $\mathcal{X}$ itself, that is when the model cannot be generalized.

V. BREAKING SYMMETRIES IN MAX#SAT

Symmetries are a special kind of permutations of the input variables of a formula leaving it intact. Exploiting or breaking symmetries in SAT formulas has long been a topic of interest.

For instance, if a formula is left intact by such a permutation then for each blocking clause C, the solver may need to generate the full orbit of C by the group of permutations, leading to combinatorial explosion. Breaking the symmetry means selecting one solution per orbit by adding a predicate called *symmetry breaking predicate* to the formula, purposefully generated to break the symmetries. The resulting formula is equisatisfiable, but often simpler to solve.

A. Correctness in the presence of symmetries

In our context, handling symmetries within the witness set reduces the size of the search space, and leads to better complexity. We give in this section arguments about why this is true.

Definition V.1. Given a Boolean formula $\phi(\mathcal{X},\mathcal{Y},\mathcal{Z})$, a *symmetry* of ϕ is a bijective function $\sigma : \overline{\mathcal{X}} \mapsto \overline{\mathcal{X}}$ that preserves negation, that is $\sigma(\neg X) = \neg\sigma(X)$, and such that, when σ is lifted to formulas, $\sigma(\phi) = \phi$ syntactically [11].

S_ϕ denotes the set of all symmetries of ϕ. We lift S_ϕ to models by defining the set of symmetries of a model x, $S_\phi(x) = \{x \circ \sigma \mid \sigma \in S_\phi\}$.

Theorem V.1. *In Algorithm 1, picking only one x per symmetry class of ϕ preserves the correctness of the algorithm both in the exact and approximate case.*

Proof. Whatever the method used to select only one member of each symmetry class, this corresponds to creating a symmetry breaking predicate $\psi(\mathcal{X})$ and solving the problem over $\phi \wedge \psi$, and thus the Property II.5 applies. □

B. Implementing Max#SAT symmetry breaking

We detect symmetries in ϕ using the automorphisms of a colored graph representing the formula, defined as follows:

- For each variable, create two nodes: one for the positive literal, and one for the negative literal. Use color 0 if the variable is in X, otherwise use color 1. Add an edge (*Boolean consistency edge*) between the two nodes.
- For each clause, create a node, and assign to it the color 2. Add an edge between this clause node and every node corresponding to a literal present in the clause.

Many tools can be used in order to list the automorphisms of a graph. In our case, we used BLISS [12] because of its C++ interface, and its performance.

After detecting the symmetries, one can use any symmetry breaking technique available, either static [13] or dynamic [6]. In our implementation, we chose to use CDCLSYM [6] because of its ease of use, and because it avoids generating complex symmetry breaking predicates ahead of time.

VI. HEURISTICS AND OPTIMIZATIONS

We present in this section heuristics used in both Algorithms 1 and 2 in practice, and discuss their effectiveness.

A. Progressive construction of the candidate

A simple yet effective optimization is to gradually add literals to the candidate x in Algorithm 1 at Line 7. By stopping earlier, this allows to call GENERALIZE on a partial assignment instead of a complete one, and will decrease the number of calls to the #SAT oracle as it anticipates work that is done in Algorithm 2.

B. Leads

When performing the generalization in Algorithm 2, one can see that we can extract *hints* about promising parts of the search space when relaxing variables. Indeed, when relaxing parts of the solution (Lines 16 and 22), if the model count of the relaxation goes above n_m, then this part of the search space may contain an improvement over the current solution.

Following this intuition, one can hold a *sorted list*[1] of relaxations whose count is above the current best known maximum, and use it to favor parts of the search space that look promising. We call these promising relaxations *leads*. More formally, given $\tilde{x}|_{\mathcal{E}}$ a lead, when searching for a new solution in Algorithm 1 at Line 7, instead of searching in $\mathcal{M}_{\mathcal{X}}(\phi_s)$, one would search in $\mathcal{M}_{\mathcal{X}}(\phi_s) \cap [\tilde{x}]_{\mathcal{E}}$.

Let $L_n(\phi)$ denote the set of leads currently known to the solver with count lower than n. When the currently known maximum is improved in Algorithm 1 at Line 10, we can block all leads whose count is below the new maximum:

$$\phi_{si+1} = \phi_{si} \wedge \bigwedge_{[\tilde{x}]_{\mathcal{E}} \in L_{n_m}(\phi)} \neg\left(\tilde{x}|_{\mathcal{E}}\right)$$

[1]The order to use here is: first the count of the relaxation, then the size of the relaxation.

C. Decision heuristic

As discussed in Section IV-A, the performances of the algorithm depend on the order with which variables are considered in various parts of the solving process (in the generalization and during the optimization presented in Section VI-A). One can see that this kind of problem, that we call *variable scheduling*, is actually predominant when solving SAT problems, and even #SAT problems.

One first heuristic arises from the leads described in Section VI-B. One can use the leads list as indications for literals leading to promising parts of the search space, by finding the literal which appears the most in the leads. We call this heuristic `leads`.

Another decision heuristic can be devised using VSIDS [14]. The idea is to assign a weight to each literal based on its last appearance in a blocking clause. The weight of each literal is increased by a constant amount every time the literal appears in a blocking clause, and is multiplicatively decreased at each blocking clause. This heuristic showed promising results in both SAT and #SAT [15]. We call this heuristic `vsids`.

One could also choose the next decision variable at random, which we call `rnd`. And finally, one could just pick the decision variables in the order they are provided to the tool, which we call `none`.

An experimental evaluation is done in Section VII-B.

D. Handling equivalent literals

Equivalent literals are a notorious property of Boolean formulas which, when exploited, results generally in better runtime performances [16].

Definition VI.1. Given a Boolean formula ϕ, we say that two literals $L_i \in \overline{\mathcal{V}}$ and $L_j \in \overline{\mathcal{V}}$ are *equivalent* if $\phi \models L_i \Leftrightarrow L_j$.

Equivalent literals allow to simplify formulas based on the following theorem.

Theorem VI.1. *Let ϕ be a Boolean formula and two equivalent literals L_i and L_j. Then solving the Max#SAT problem for ϕ is reduced to solving the Max#SAT problem for the simpler formula ϕ' obtained by replacing all occurrences of L_j (resp. $\neg L_j$) by L_i (resp. $\neg L_i$) when:*

1) *either L_i and L_j are in the same literal class (either $\overline{\mathcal{X}}$, $\overline{\mathcal{Y}}$ or $\overline{\mathcal{Z}}$)*
2) *or $L_i \in \overline{\mathcal{X}}$ and $L_j \in \overline{\mathcal{Y}} \cup \overline{\mathcal{Z}}$*
3) *or $L_i \in \overline{\mathcal{Y}}$ and $L_j \in \overline{\mathcal{Z}}$.*

Theorem VI.1 can be applied multiple times in order to further simplify the formula. Literal equivalence can be detected using binary implication graphs [17].

VII. Experimental evaluation

Algorithm 1 has been implemented in an open-source tool written in C++ called BAXMC [7], including dynamic symmetry breaking techniques (Section V) and all the heuristics discussed in Section VI. In this implementation, we only incorporated the approximate version of the algorithm using APPROXMC5 [18] as an approximated model counting oracle and CRYPTOMINISAT [19] as a SAT solver oracle. An exact solver is not implemented because we do not, at the time of writing, have another exact Max#SAT solver available as a comparison.

We use three sets of benchmarks, coming either from [20], or from MaxSat 2021 competition [21]. Benchmarks from this later class are transformed using the method from [1]. Table I shows more details about the benchmark set considered. Benchmarks annoted with a star indicate that a symmetry was found.

All experiments are run on a Dell R640 with 40 cores and 192 GB of RAM running Debian 11, with a 2-hour timeout, a 10 GB memory limit and with parameters $\delta = 0.2$, $\epsilon = 0.8$.

A. Comparison to MAXCOUNT

MAXCOUNT [1] is used as an off-the-shelf solver of the problem, with parameters corresponding to $\delta = 0.2$, $\epsilon = 0.8$. Note that these are not the parameters used in the experiments in [1] and that we reimplemented MAXCOUNT using newer oracles. We did this in order to see how MAXCOUNT and BAXMC behave when both are providing the same correctness guarantees and using the same oracles for fairness. All figures from Table II are obtained when BAXMC is used with the `(leads,rnd)` heuristic combination.

Table II shows the results obtained when running both tools on our three benchmarks. Bolded values are the *best* values on this line (i.e., smaller time or biggest answer). The *time* columns are the running times of the tools. The *model count* columns are the values returned by the candidate tools.

One can see that BAXMC outperforms MAXCOUNT in all benchmark timings. In cases where BAXMC did not find the best value, it terminates when the bounds on the possible maximum are *tight enough*. This yields a small error margin on the returned value of BAXMC, but is configurable through its κ argument.

B. Decision heuristic comparison

Table III shows a comparison between the heuristics that are currently available in BAXMC. Lines enumerate the decision heuristics from Section VI-C. Columns specify heuristics used by the underlying SAT oracle about literals polarities.

Each cell of this table contains, in sequence: the total running time, the number of time this combination ran the fastest compared to all others, and the number of times this combination timed out. For example combination `(leads,cache)` ran for a total time of 62968.38 seconds with 7 timeouts, and ran the fastest on 3 benchmarks over a total number of 26. In this setup, any time-out from BAXMC increases the total running time by 7200s.

The table shows that none of the heuristics stands out. We can only eliminate random decision as a bad heuristic. Nevertheless, the combination of heuristics allows to strongly reduce the overall number of timeouts.

VIII. Related works

Previous works on Max#SAT solving may be classified into three categories, based respectively on probabilistic solving as in MAXCOUNT [1], exhaustive search [5] and knowledge compilation [22].

Probabilistic solving relies on "amplification" to build a new formula $\tilde{\phi}(\mathcal{X}, \mathcal{Y}, \mathcal{Z}) = \bigwedge_{i=1}^{k} \phi(\mathcal{X}, \mathcal{Y}_i, \mathcal{Z}_i)$, where the $\mathcal{Y}_i$ and $\mathcal{Z}_i$ are fresh copies of the initial $\mathcal{Y}$ and $\mathcal{Z}$ variables, and uniformly sampling among $\mathcal{M}_{\mathcal{X}}\left(\tilde{\phi}\right)$. The higher the k, the more the sampling is attracted towards the $\mathcal{X}$ with large projected model counting over $(\mathcal{Y}_i)_{i \leq k}$. Given parameters ϵ and δ, the guarantees provided about the returned tuple $(\tilde{n}, \tilde{x})$ are the same as in Corollary III.1 [1]. Unfortunately, when the size of the formula increases, uniform sampling may become quite expensive as shown in our benchmarks. Furthermore, this approach is not incremental: looking for a better solution involves re-running the search from scratch.

On the other side of the spectrum lie *exhaustive searches*. The idea here is to make incremental decisions among the variables in $\mathcal{X}$, propagating the decision in ϕ, and simplifying the formula in order to cache some results [5]. Such approaches are exact, but their exhaustive nature limits their scalability. Component caching [23] is a practical way to improve scalability [5] and it could be beneficial into our algorithm too.

Knowledge compilation consists in compiling the formula into a representation over which solving the problem (here, the optimal model counting) is expected to be much easier. Compilation times tend to dominate and the memory usage of the compiled form may be huge.

A possible approach could use a generalization of $\mathcal{X}$-constrained SDDs [22]. The idea here would be to build $(\mathcal{X}, \mathcal{Y})$-constrained SDDs, that is SDDs that are $\mathcal{X}$-constrained, and for which each subtree that are not over $\mathcal{X}$ are $\mathcal{Y}$-constrained. In this case, one can easily compute the count of every possible pair $x \in \mathcal{M}_{\mathcal{X}}(\phi)$ and then propagate the maximum to the root of the tree. To the best of our knowledge, this direction has not been explored yet.

IX. Conclusion and future work

We proposed a CEGAR based algorithm allowing to solve medium-sized instances of the Max#SAT within reasonable time limits, as illustrated in our experiments. This algorithm allows either to compute exact solutions (when possible), or can be smoothly relaxed to produce approximated results, under well-defined probabilistic guarantees. Comparisons with an existing probabilistic tool showed the gains provided by our algorithm on concrete examples. Our implementation and all the related benchmarks are available on [7].

From an algorithmic point of view this work could be extended in several directions.

Table I
BENCHMARK LIST

Name	$\|\mathcal{X}\|$	$\|\mathcal{Y}\|$	$\|\mathcal{Z}\|$	Nr. Clauses
backdoor-32-24*	32	32	83	76
backdoor-2x16-8*	32	32	136	272
pwd-backdoor	64	64	272	609
bin-search-16	16	16	1416	5825
CVE-2007-2875	32	32	720	1740
CVE-2009-3002	288	240	443	180
reverse	32	32	165	293
ActivityService	70	34	4063	15257
ActivityService2	70	34	4063	15257
ConcreteActivityService	71	37	4728	17856
GuidanceService	69	27	3167	11612
GuidanceService2	69	27	3167	11612
IssueServiceImpl	77	29	3519	13024
IterationService	70	34	4063	15257
LoginService	92	27	5110	21559
NotificationServiceImpl2	87	32	5223	22006
PhaseService	70	34	4063	15257
ProcessBean	166	39	9675	41444
ProjectService	134	48	6778	24944
sign	16	16	107	392
sign_correct	16	16	92	346
UserServiceImpl	87	31	3901	14653
drmx	1030	17	26	2094
keller4	43	15	62	2525
g2_n35e34_n58e61	34	7	954	38130

Table II
PERFORMANCE COMPARISONS BETWEEN BAXMC AND MAXCOUNT

Benchmark name	BAXMC			MAXCOUNT	
	Time (s)	Sym. Time (s)	Model count (log)	Time (s)	Model count (log)
backdoor-32-24*	611.12	**34.50**	**32**	231.87	**32**
backdoor-2x16-8*	**60.02**	61.07	**16**	6512.28	**16**
pwd-backdoor	**236.87**	240.63	**64**	TO	-
bin-search-16	1067.38	**1048.43**	**16**	1490.44	**16**
CVE-2007-2875	**36.14**	37.39	**32**	TO	-
CVE-2009-3002	TO	TO	-	MO	-
reverse	TO	TO	-	MO	-
ActivityService	**3060.39**	3064.60	**33.95**	TO	-
ActivityService2	3096.54	**2999.72**	**33.95**	TO	-
ConcreteActivityService	**84.20**	84.44	**36.91**	TO	-
GuidanceService	**1468.39**	1474.51	**26.88**	TO	-
GuidanceService2	**1459.74**	1474.76	**26.88**	TO	-
IssueServiceImpl	1603.21	**1583.50**	**28.88**	TO	-
IterationService	3081.86	**3068.95**	**33.95**	TO	-
LoginService	5275.25	**5197.84**	**26.92**	TO	-
NotificationServiceImpl2	**1286.48**	1287.94	**31.91**	TO	-
PhaseService	**3071.86**	3105.18	**33.95**	TO	-
ProcessBean	TO	TO	-	TO	-
ProjectService	5770.26	**5544.02**	**47.92**	TO	-
sign	73.56	**73.43**	15.90	819.58	**16**
sign_correct	74.58	**73.78**	15.89	819.56	**16**
UserServiceImpl	TO	TO	-	TO	-
drmx	24.39	**24.07**	**16.99**	TO	-
keller4	TO	TO	-	TO	-
g2_n35e34_n58e61	**0.17**	0.41	**2.53**	TO	-

Table III
PERFORMANCE COMPARISON BETWEEN HEURISTICS OF BAXMC

	cache	neg	pos	rnd
leads	62968.38 – **3** – 7	65542.94 – 2 – 6	65222.40 – 1 – **4**	67233.96 – 0 – 5
rnd	144081.73 – 0 – 19	139002.15 – 0 – 18	140755.04 – 0 – 17	137407.70 – 0 – 17
none	60368.28 – **3** – 5	62729.76 – 2 – **4**	61317.54 – **3** – **4**	56860.50 – **3** – **4**
vsids	69165.19 – 2 – 8	56189.26 – 1 – 5	**54017.07** – **3** – **4**	63865.03 – 2 – 6

First, we exploited some classes of symmetries when solving Max#SAT (Section V). This could be improved by detecting new kinds of symmetries [13], or exploiting them further using techniques such as symmetry propagation [24].

As discussed in Section IV, our relaxation algorithm (Algorithm 2) uses a *linear sweep* over the literals composing a witness. Instead of returning one possible minimal relaxation, MERGEXPLAIN [25] returns multiple ones, which may be helpful in our case by allowing the creation of multiple blocking clauses.

As expected, in some instances, our algorithm may degenerate into exhaustive search. While we do not know yet any characterization of all such instances, we believe that preprocessing and in-processing [26] techniques such as UNHIDING [17] should improve performances and limit the set of inefficient instances.

Finally, Algorithm 1 may be parallelized by correctly scheduling search spaces among threads, possibly using the leads described in Section VI-B. If we enforce the fact that all leads currently present in the lead list are disjoint, that is the $[\tilde{x}]_{\mathcal{E}}$ are pairwise disjoint (hence splitting the search space into parts), we expect a favorable parallelization setting.

REFERENCES

[1] D. Fremont, M. Rabe, and S. Seshia, "Maximum model counting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[2] S. Saha, W. Eiers, I. B. Kadron, L. Bang, and T. Bultan, "Incremental attack synthesis," *ACM SIGSOFT Software Engineering Notes*, vol. 44, no. 4, pp. 16–16, 2021.

[3] D. Monniaux, "$\mathsf{NP}^{\#\mathsf{P}} = \exists\mathsf{PP}$ and other remarks about maximized counting," https://hal.archives-ouvertes.fr/hal-03586193, Feb. 2022.

[4] J. Torán, "Complexity classes defined by counting quantifiers," *J. ACM*, vol. 38, no. 3, pp. 753–774, 1991.

[5] G. Audemard, J.-M. Lagniez, M. Miceli, and O. Roussel, "Identifying soft cores in propositional formulæ," 2022.

[6] H. Metin, S. Baarir, M. Colange, and F. Kordon, "Cdclsym: Introducing effective symmetry breaking in sat solving," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2018, pp. 99–114.

[7] "Baxmc website." [Online]. Available: https://www-verimag.imag.fr/~vigourth/research/baxmc/

[8] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "A scalable approximate model counter," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2013, pp. 200–216.

[9] U. Junker, "QuickXplain: Conflict detection for arbitrary constraint propagation algorithms," in *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, vol. 4. Citeseer, 2001.

[11] H. Zhang, "Combinatorial designs by sat solvers 1," in *Handbook of Satisfiability*. IOS Press, 2021, pp. 819–858.

[10] D. Monniaux, "Quantifier elimination by lazy model enumeration," in *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, ser. Lecture Notes in Computer Science, T. Touili, B. Cook, and P. B. Jackson, Eds., vol. 6174. Springer, 2010, pp. 585–599.

[12] T. Junttila and P. Kaski, "Engineering an efficient canonical labeling tool for large and sparse graphs," in *2007 Proceedings of the Ninth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2007, pp. 135–149.

[13] J. Devriendt, B. Bogaerts, M. Bruynooghe, and M. Denecker, "Improved static symmetry breaking for sat," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2016, pp. 104–122.

[14] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th annual Design Automation Conference*, 2001, pp. 530–535.

[15] T. Sang, P. Beame, and H. Kautz, "Heuristics for fast exact model counting," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2005, pp. 226–240.

[16] Y. Lai, K. S. Meel, and R. H. Yap, "The power of literal equivalence in model counting," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 3851–3859.

[17] M. J. Heule, M. Järvisalo, and A. Biere, "Efficient cnf simplification based on binary implication graphs," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2011, pp. 201–215.

[18] K. S. Meel and S. Akshay, "Sparse hashing for scalable approximate model counting: theory and practice," in *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, 2020, pp. 728–741.

[19] M. Soos, K. Nohl, and C. Castelluccia, "Extending SAT solvers to cryptographic problems," in *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, ser. Lecture Notes in Computer Science, O. Kullmann, Ed., vol. 5584. Springer, 2009, pp. 244–257. [Online]. Available: https://doi.org/10.1007/978-3-642-02777-2_24

[20] "Maxcount 1.0.0." [Online]. Available: https://github.com/dfremont/maxcount

[21] "Maxsat evaluation 2021." [Online]. Available: https://maxsat-evaluations.github.io/2021/index.html

[22] U. Oztok, A. Choi, and A. Darwiche, "Solving pp pp-complete problems using knowledge compilation," in *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2016.

[23] F. Bacchus, S. Dalmao, and T. Pitassi, "Dpll with caching: A new algorithm for #sat and bayesian inference," in *Electronic Colloquium in Computation Complexity*. Citeseer, 2003.

[24] H. Metin, S. Baarir, and F. Kordon, "Composing symmetry propagation and effective symmetry breaking for sat solving," in *NASA Formal Methods Symposium*. Springer, 2019, pp. 316–332.

[25] K. Shchekotykhin, D. Jannach, and T. Schmitz, "Mergexplain: Fast computation of multiple conflicts for diagnosis," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[26] M. Järvisalo, M. J. Heule, and A. Biere, "Inprocessing rules," in *International Joint Conference on Automated Reasoning*. Springer, 2012, pp. 355–370.