Query-Guided Maximum Satisfiability

Xin Zhang[†] Ravi Mangal[†] Aditya V. Nori[*] Mayur Naik[†]

[†] Georgia Institute of Technology, USA [*] Microsoft Research, UK

xin.zhang@gatech.edu ravi.mangal@gatech.edu adityan@microsoft.com naik@cc.gatech.edu

Abstract

We propose a new optimization problem "Q-MAXSAT", an extension of the well-known Maximum Satisfiability or MAXSAT problem. In contrast to MAXSAT, which aims to find an assignment to *all* variables in the formula, Q-MAXSAT computes an assignment to a desired *subset* of variables (or queries) in the formula. Indeed, many problems in diverse domains such as program reasoning, information retrieval, and mathematical optimization can be naturally encoded as Q-MAXSAT instances.

We describe an iterative algorithm for solving Q-MAXSAT. In each iteration, the algorithm solves a subproblem that is relevant to the queries, and applies a novel technique to check whether the partial assignment found is a solution to the Q-MAXSAT problem. If the check fails, the algorithm grows the subproblem with a new set of clauses identified as relevant to the queries.

Our empirical evaluation shows that our Q-MAXSAT solver PILOT achieves significant improvements in runtime and memory consumption over conventional MAXSAT solvers on several Q-MAXSAT instances generated from real-world problems in program analysis and information retrieval.

Categories and Subject Descriptors G.1.6 [*optimization*]: constrained optimization

General Terms Algorithms, Theory

Keywords Optimization, Maximum Satisfiability, Partial Model, Query-Guided Approach, Program Analysis, Information Retrieval

1. Introduction

The maximum satisfiability or MAXSAT problem [47] is an optimization extension of the well-known satisfiability or SAT problem [13, 17]. A MAXSAT formula consists of a set of conventional SAT or hard clauses together with a set of soft or weighted clauses. The solution to a MAXSAT formula is an assignment to its variables that satisfies all the hard clauses, and maximizes the sum of the weights of satisfied soft clauses. A number of interesting problems that span across a diverse set of areas such as program reasoning [24, 25, 35, 58], information retrieval [14, 29, 45, 48], databases [5, 40], circuit design [56], bioinformatics [16, 52], planning and scheduling [26, 53, 57], and many others can be naturally encoded as MAXSAT formulae. Many of these problems specify hard constraints that encode various soundness conditions, and soft constraints that specify objectives to optimize.

MAXSAT solvers have made remarkable strides in performance over the last decade [4, 8, 20, 22, 37, 39, 41, 42, 44]. This in turn has motivated even more demanding and emerging applications to be formulated using MAXSAT. Real-world instances of many of these problems, however, result in very large MAXSAT formulae [36]. These formulae, comprising millions of clauses or more, are beyond the scope of existing MAXSAT solvers. Fortunately, for many of these problems, one is interested in a small set of *queries* that constitute a very small fraction of the entire MAXSAT solution. For instance, in program analysis, a query could be analysis information for a particular variable in the program—intuitively, one would expect the computational cost for answering a small set of queries to be much smaller than the cost of computing analysis information for all program variables. In the MAXSAT setting, the notion of a query translates to the value of a specific variable in a MAXSAT solution. Given a MAXSAT formula φ and a set of queries $\mathcal{Q}$, one obvious method for answering queries in $\mathcal{Q}$ is to compute the MAXSAT solution to φ and project it to variables in $\mathcal{Q}$. Needless to say, especially for very large MAXSAT formulae, this is a non-scalable solution. Therefore, it is interesting to ask the following question: "*Given a* MAXSAT *formula φ and a set of queries $\mathcal{Q}$, is it possible to answer $\mathcal{Q}$ by only computing information relevant to $\mathcal{Q}$?*". We call this question the *query-guided maximum satisfiability* or Q-MAXSAT problem (φ, $\mathcal{Q}$).

Our main technical insight is that a Q-MAXSAT instance (φ, $\mathcal{Q}$) can be solved by computing a MAXSAT solution of a small subset of the clauses in φ. The main challenge, however, is how to efficiently determine whether the answers to $\mathcal{Q}$ indeed correspond to a MAXSAT solution of φ. We propose an iterative algorithm for solving a Q-MAXSAT instance (φ, $\mathcal{Q}$) (Algorithm 1 in Section 4). In each iteration, the algorithm computes a MAXSAT solution to a subset of clauses in φ that are relevant to $\mathcal{Q}$. We also define an algorithm (Algorithm 2 in Section 4.1) that efficiently checks whether the current assignment to variables in $\mathcal{Q}$ corresponds to a MAXSAT solution to φ. In particular, our algorithm constructs a small set of clauses that succinctly summarize the effect of the clauses in φ that are not considered by our algorithm, and then uses it to overestimate the gap between the optimum objective value of φ under the current assignment and the optimum objective value of φ.

We have implemented our approach in a tool called PILOT and applied it to 19 large MAXSAT instances ranging in size from 100 thousand to 22 million clauses generated from real-world problems in program analysis and information retrieval. Our empirical evaluation shows that PILOT achieves significant improvements in runtime and memory over conventional MAXSAT solvers: on these instances, PILOT used only 285 MB of memory on average and terminated in 107 seconds on average. In contrast, conventional MAXSAT solvers timed out for eight of the instances.

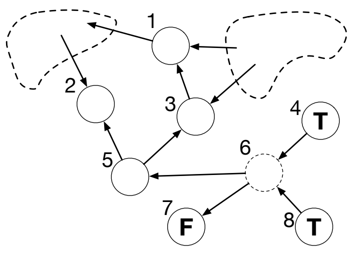


Figure 1: Graph representation of a large MAXSAT formula φ.

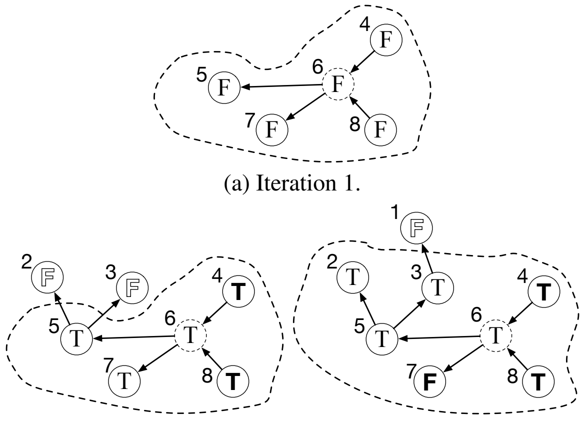


Figure 2: Graph representation of each iteration in our algorithm when it solves the Q-MAXSAT instance $(\varphi, \{v_6\})$.

In summary, the main contributions of this paper are as follows:

1. We introduce and formalize a new optimization problem called Q-MAXSAT. In contrast to traditional MAXSAT, where one is interested in an assignment to all variables, for Q-MAXSAT, we are interested only in an assignment to a subset of variables, called queries.
2. We propose an iterative algorithm for Q-MAXSAT that has the desirable property of being able to efficiently check whether an assignment to queries is an optimal solution to Q-MAXSAT.
3. We present empirical results showing the effectiveness of our approach by applying PILOT to several Q-MAXSAT instances generated from real-world problems in program analysis and information retrieval.

2. Example

We illustrate the Q-MAXSAT instance and our solution with the help of an example. Figure 1 represents a large MAXSAT formula φ in conjunctive form. Each variable v_i in φ is represented as a node in the graph labeled by its subscript i. Each clause is a disjunction of literals with a positive weight. Nodes marked as **T** (or **F**) indicate soft clauses of the form $(100, v_i)$ (or $(100, \neg v_i)$) while each edge from node i to a node j represents a soft clause of the form $(5, \neg v_i \vee v_j)$.

Suppose we are interested in the assignment to the query v_6 in φ (shown by the dashed node in Figure 1). Then, the Q-MAXSAT instance that we want to solve is $(\varphi, \{v_6\})$. A solution to this Q-MAXSAT instance is a partial model which maps v_6 to *true* or *false* such that there is a completion of this partial model maximizing the objective value of φ.

A naive approach to solve this problem is to directly feed φ into a MAXSAT solver and extract the assignment to v_6 from the solution. However, this approach is highly inefficient due to the large size of practical instances.

Our approach exploits the fact that the Q-MAXSAT instance only requires assignment to specific query variables, and solves the problem lazily in an iterative manner. The high level strategy of our approach (see Section 4 for details) is to only solve a subset of clauses relevant to the query in each iteration, and terminate when the current solution can be proven to be the solution to the Q-MAXSAT instance. In particular, our approach proceeds in the following four steps.

Initialize. Given a Q-MAXSAT instance, our query-guided approach constructs an initial set of clauses that includes all the clauses containing a query variable. We refer to these relevant clauses as the *workset* of our algorithm.

Solve. The next step is to solve the MAXSAT instance induced by the workset. Our approach uses an existing off-the-shelf MAXSAT solver for this purpose. This produces a partial model of the original instance.

Check. The key step in our approach is to check whether the current partial model can be completed to a model of the original MAXSAT instance. Since the workset only includes a small subset of clauses from the complete instance, the challenge here is to summarize the effect of the unexplored clauses on the assignment to the query variables. We propose a novel technique for performing this check efficiently without actually considering all the unexplored clauses (see Section 4.1 for formal details).

Expand. If the check in the previous step fails, it indicates that we need to grow our workset of relevant clauses. Based on the check failure, in this step, our approach identifies the set of clauses to be added to the workset for the next iteration.

As long as the Q-MAXSAT instance is finite, our iterative approach is guaranteed to terminate since it only grows the workset in each iteration.

We next describe how our approach solves the Q-MAXSAT instance $(\varphi, \{v_6\})$. To resolve v_6, our approach initially constructs the workset φ' that includes all the clauses containing v_6. We represent φ' by the subgraph contained within the dotted area in Figure 2(a). By invoking a MAXSAT solver on φ', we get a partial model $\alpha_{\varphi'} = [v_4 \mapsto \mathit{false}, v_5 \mapsto \mathit{false}, v_6 \mapsto \mathit{false}, v_7 \mapsto \mathit{false}, v_8 \mapsto \mathit{false}]$ as shown in Figure 2(a), with an objective value of 20. Our approach next checks if the current partial model found is a solution to the Q-MAXSAT instance $(\varphi, \{v_6\})$. It constructs a set of clauses ψ which succinctly summarizes the effect of the clauses that are not present in the workset. We refer to this set as the *summarization set*, and use the following expression to overestimate the gap between the optimum objective value of φ under the current partial model and the optimum objective value of φ:

$$\max_{\alpha} \mathit{eval}(\varphi' \cup \psi, \alpha) - \max_{\alpha} \mathit{eval}(\varphi', \alpha),$$

where $\max_\alpha \mathit{eval}(\varphi' \cup \psi, \alpha)$ and $\max_\alpha \mathit{eval}(\varphi', \alpha)$ are the optimum objective values of $\varphi' \cup \psi$ and φ', respectively.

To construct such a summarization set, our insight is that the clauses that are not present in the workset can only affect the query assignment via clauses sharing variables with the workset. We call such clauses as the *frontier clauses*. Furthermore, if a frontier clause is already satisfied by the current partial model, expanding the workset with such a clause cannot further improve the partial model. We now try to construct a summarization set ψ by taking all frontier clauses not satisfied by $\alpha_{\varphi'}$. As a result, our algorithm produces $\psi = \{(100, v_4), (100, v_8)\}$. The check comparing the optimum objective values of φ' and $\varphi' \cup \psi$ fails in this case. In particular, by solving $\varphi' \cup \psi$, we get a partial model $\alpha_{\varphi' \cup \psi} = [v_4 \mapsto \mathit{true}, v_5 \mapsto \mathit{true}, v_6 \mapsto \mathit{true}, v_7 \mapsto \mathit{true}, v_8 \mapsto \mathit{true}]$ with 220 as the objective value, which is greater than the optimum objective value of φ'. As a consequence, our approach expands the

$$\begin{array}{rrcl}
(\textit{variable}) & v & \in & \mathcal{V} \\
(\textit{clause}) & c & ::= & \bigvee_{i=1}^{n} v_i \vee \bigvee_{j=1}^{m} \neg v'_j \ \mid\ 1 \ \mid\ 0 \\
(\textit{weight}) & w & \in & \mathbb{R}^{+} \\
(\textit{soft clause}) & s & ::= & (w, c) \\
(\textit{formula}) & \varphi & ::= & \{s_1, ..., s_n\} \\
(\textit{model}) & \alpha & \in & \mathcal{V} \to \{0,1\}
\end{array}$$

$$\mathit{fst} = \lambda(w,c).w, \quad \mathit{snd} = \lambda(w,c).c$$

$$\forall \alpha : \alpha \models 1, \quad \forall \alpha : \alpha \not\models 0$$

$$\begin{array}{rcl}
\alpha \models \bigvee_{i=1}^{n} v_i \vee \bigvee_{j=1}^{m} \neg v'_j & \Leftrightarrow & \exists i : \alpha(v_i) = 1 \text{ or } \exists j : \alpha(v'_j) = 0 \\
\mathit{eval}(\{s_1, ..., s_n\}, \alpha) & = & \Sigma_i^n \{\mathit{fst}(s_i) \mid \alpha \models \mathit{snd}(s_i)\} \\
\mathit{MaxSAT}(\varphi) & = & \mathrm{argmax}_\alpha\ \mathit{eval}(\varphi, \alpha)
\end{array}$$

Figure 3: Syntax and interpretation of MAXSAT formulae.

Algorithm 1

```
 1: INPUT: A Q-MAXSAT instance $(\varphi, \mathcal{Q})$, where $\varphi$ is a MAXSAT
    formula, and $\mathcal{Q}$ is a set of queries.
 2: OUTPUT: A solution to the Q-MAXSAT instance $(\varphi, \mathcal{Q})$.
 3: $\varphi' :=$ INIT$(\varphi, \mathcal{Q})$
 4: while true do
 5:     $\alpha_{\varphi'} :=$ MAXSAT$(\varphi')$
 6:     $\varphi'' :=$ CHECK$((\varphi, \mathcal{Q}), \varphi', \alpha_{\varphi'})$, where $\varphi'' \subseteq \varphi \setminus \varphi'$
 7:     if $\varphi'' = \emptyset$ then
 8:         return $\lambda v.\alpha_{\varphi'}(v)$, where $v \in \mathcal{Q}$
 9:     else
10:         $\varphi' := \varphi' \cup \varphi''$
11:     end if
12: end while
```

workset φ' with $(100, v_4)$ and $(100, v_8)$ and proceeds to the next iteration. We include these two clauses as these are not satisfied by $\alpha_{\varphi'}$ in the last iteration, but satisfied by $\alpha_{\varphi' \cup \psi}$, which indicates they are likely responsible for the failure of the previous check.

In iteration 2, invoking a MAXSAT solver on the new workset, φ', produces $\alpha_{\varphi'} = [v_4 \mapsto \mathit{true}, v_5 \mapsto \mathit{true}, v_6 \mapsto \mathit{true}, v_7 \mapsto \mathit{true}, v_8 \mapsto \mathit{true}]$, as shown in Figure 2(b), with 220 as the objective value. The violated frontier clauses in this case are $(100, \neg v_7)$, $(5, \neg v_5 \vee v_2)$, and $(5, \neg v_5 \vee v_3)$. However, if this set of violated frontier clauses were to be used as the summarization set ψ, the newly added variables v_2 and v_3 will cause the check comparing the optimum objective values of φ' and $\varphi' \cup \psi$ to trivially fail. To address this problem, and further improve the precision of our check, we modify the summarization set as $\psi = \{(100, \neg v_7)$, $(5, \neg v_5)$, $(5, \neg v_5)\}$ by removing v_2 and v_3 and strengthening the corresponding clauses. In this case, it is equivalent to setting v_2 and v_3 to *false* (marked as $\mathbb{F}$ in the graph). Intuitively, by setting these variables to *false*, we are overestimating the effects of the unexplored clauses, by assuming these frontier clauses will not be satisfied by unexplored variables like v_2 and v_3. By solving $\varphi' \cup \psi$, we get a partial model $\alpha_{\varphi' \cup \psi} = [v_4 \mapsto \mathit{true}, v_5 \mapsto \mathit{false}, v_6 \mapsto \mathit{true}, v_7 \mapsto \mathit{false}, v_8 \mapsto \mathit{true}]$ with 320 as the objective value, which is greater than the optimum objective value of φ'.

In iteration 3, as shown in Figure 2(c), our approach expands the workset φ' with $(100, \neg v_7)$, $(5, \neg v_5 \vee v_2)$ and $(5, \neg v_5 \vee v_3)$. By invoking a MAXSAT solver on φ', we produce $\alpha_{\varphi'} = [v_2 \mapsto \mathit{true}, v_3 \mapsto \mathit{true}, v_4 \mapsto \mathit{true}, v_5 \mapsto \mathit{true}, v_6 \mapsto \mathit{true}, v_7 \mapsto \mathit{false}, v_8 \mapsto \mathit{true}]$ with an objective value of 325. We omit the edges representing frontiers clauses that are already satisfied by $\alpha_{\varphi'}$ in Figure 2(c). To check the whether current partial model is a solution to the Q-MAXSAT instance, we construct strengthened summarization set $\psi = \{(5, \neg v_3)\}$ from the frontier clause $(5, \neg v_3 \vee v_1)$. By invoking MAXSAT on $\varphi' \cup \psi$, we get an optimum objective value of 325 which is the same as that of φ'. As a result, our algorithm terminates and extracts $[v_6 \mapsto \mathit{true}]$ as the solution to the Q-MAXSAT instance.

Despite the fact that many clauses and variables can reach v_6 in the graph (i.e, they might affect the assignment to the query), our approach successfully resolves v_6 in three iterations and only explores a very small, local subset of the graph, while successfully summarizing the effects of the unexplored clauses.

3. The Q-MAXSAT Problem

First, we describe the standard MAXSAT problem [47]. The syntax of a MAXSAT formula is shown in Figure 3. A MAXSAT formula φ consists of a set of *soft clauses*. Each soft clause $s = (w, c)$ is a pair that consists of a positive weight $w \in \mathbb{R}^{+}$, and a clause c that is a disjunctive form over a set of variables $\mathcal{V}$ [1]. We use 1 to denote *true* and 0 to denote *false*. Given an assignment $\alpha : \mathcal{V} \to \{0, 1\}$ to each variable in a MAXSAT formula φ, we use $\mathit{eval}(\varphi, \alpha)$ to denote the sum of the weights of the soft clauses in the formula that are satisfied by the assignment. We call this sum the *objective value* of the formula under that assignment. The space of *models* $\mathit{MaxSAT}(\varphi)$ of a MAXSAT formula φ is the set of all assignments that maximize the objective value of φ.

The query-guided maximum satisfiability or Q-MAXSAT problem is an extension to the MAXSAT problem, that augments the MAXSAT formula with a set of *queries* $\mathcal{Q} \subseteq \mathcal{V}$. In contrast to the MAXSAT problem, where the objective is to find an assignment to all variables $\mathcal{V}$, Q-MAXSAT only aims to find a *partial model* $\alpha_{\mathcal{Q}} : \mathcal{Q} \to \{0, 1\}$. In particular, given a MAXSAT formula φ and a set of queries $\mathcal{Q}$, the Q-MAXSAT problem seeks a partial model $\alpha_{\mathcal{Q}} : \mathcal{Q} \to \{0, 1\}$ for φ, that is, an assignment to the variables in $\mathcal{Q}$ such that there exists a completion $\alpha : \mathcal{V} \to \{0, 1\}$ of $\alpha_{\mathcal{Q}}$ that is a model of the MAXSAT formula φ. Formally:

Definition 1 (Q-MAXSAT **problem).** Given a MAXSAT formula φ, and a set of queries $\mathcal{Q} \subseteq \mathcal{V}$, a model of the Q-MAXSAT instance $(\varphi, \mathcal{Q})$ is a partial model $\alpha_{\mathcal{Q}} : \mathcal{Q} \to \{0, 1\}$ such that

$$\exists \alpha \in \mathit{MaxSAT}(\varphi).(\forall v \in \mathcal{Q}.\alpha_{\mathcal{Q}}(v) = \alpha(v)).$$

Example. Let $(\varphi, \mathcal{Q})$ where $\varphi = \{(5, \neg a \vee b), (5, \neg b \vee c), (5, \neg c \vee d), (5, \neg d)\}$ and $\mathcal{Q} = \{a\}$ be a Q-MAXSAT instance. A model of this instance is given by $\alpha_{\mathcal{Q}} = [a \mapsto 0]$. Indeed, there is a completion $\alpha = [a \mapsto 0, b \mapsto 0, c \mapsto 0, d \mapsto 0]$ that belongs to $\mathit{MaxSAT}(\varphi)$ (in other words, α maximizes the objective value of φ, which is equal to 20), and agrees with $\alpha_{\mathcal{Q}}$ on the variable a (that is, $\alpha(a) = \alpha_{\mathcal{Q}}(a) = 0$). $\square$

Hereafter, we use $\mathit{Var}(\varphi)$ to denote the set of variables occurring in MAXSAT formula φ. For a set of variables $\mathcal{U} \subseteq \mathcal{V}$, we denote the partial model $\alpha : \mathcal{U} \to \{0, 1\}$ by $\alpha_{\mathcal{U}}$. Also, we use α_φ as shorthand for $\alpha_{\mathit{Var}(\varphi)}$. Throughout the paper, we assume that for $\mathit{eval}(\varphi, \alpha)$, the assignment α is well-defined for all variables in $\mathit{Var}(\varphi)$.

4. Solving a Q-MAXSAT Instance

Algorithm 1 describes our technique for solving the Q-MAXSAT problem. It takes as input a Q-MAXSAT instance $(\varphi, \mathcal{Q})$, and returns a solution to $(\varphi, \mathcal{Q})$ as output. The main idea in the algorithm is to iteratively identify and solve a subset of clauses in φ that are relevant to the set of queries $\mathcal{Q}$, which we refer to as the *workset* of our algorithm.

[1] Without loss of generality, we assume that MAXSAT formulae contain only soft clauses. Every hard clause can be converted into a soft clause with a sufficiently high weight.

Algorithm 2 CHECK$((\varphi, \mathcal{Q}), \varphi', \alpha_{\varphi'})$

```
1: INPUT: A Q-MAXSAT instance $(\varphi, \mathcal{Q})$, a MAXSAT formula
   $\varphi' \subseteq \varphi$, and a model $\alpha_{\varphi'} \in MaxSAT(\varphi')$.
2: OUTPUT: A set of clauses $\varphi'' \subseteq \varphi \setminus \varphi'$.
3: $\psi$ := APPROX$(\varphi, \varphi', \alpha_{\varphi'})$
4: $\varphi''$ := $\varphi' \cup \psi$
5: $\alpha_{\varphi''}$ := MAXSAT$(\varphi'')$
6: if $eval(\varphi'', \alpha_{\varphi''}) - eval(\varphi', \alpha_{\varphi'}) = 0$ then
7:     return $\emptyset$
8: else
9:     $\varphi_s$ := $\{(w, c) \mid (w, c) \in \psi \land \alpha_{\varphi''} \models c\}$
10:    return REFINE$(\varphi_s, \varphi, \varphi', \alpha_{\varphi'})$
11: end if
```

The algorithm starts by invoking function INIT (line 3) which returns an initial set of clauses $\varphi' \subseteq \varphi$. Specifically, φ' is the set of clauses in φ which contain variables in the query set $\mathcal{Q}$.

In each iteration (lines 4–12), Algorithm 1 first computes a model $\alpha_{\varphi'}$ of the MAXSAT formula φ' (line 5). Next, in line 6, the function CHECK checks whether the model $\alpha_{\varphi'}$ is sufficient to compute a model of the Q-MAXSAT instance $(\varphi, \mathcal{Q})$. If $\alpha_{\varphi'}$ is sufficient, then the algorithm returns a model which is $\alpha_{\varphi'}$ restricted to the variables in $\mathcal{Q}$ (line 8). Otherwise, CHECK returns a set of clauses φ'' that is added to the set φ' (line 10). It is easy to see that Algorithm 1 always terminates if φ is finite.

The main and interesting challenge is the implementation of the CHECK function so that it is sound (that is, when CHECK returns $\emptyset$, then the model $\alpha_{\varphi'}$ restricted to the variables in $\mathcal{Q}$ is indeed a model of the Q-MAXSAT instance), yet efficient.

4.1 Implementing an Efficient CHECK Function

Algorithm 2 describes our implementation of the CHECK function. The input to the CHECK function is a Q-MAXSAT instance $(\varphi, \mathcal{Q})$, a MAXSAT formula $\varphi' \subseteq \varphi$ as described in Algorithm 1, and a model $\alpha_{\varphi'} \in MaxSAT(\varphi')$. The output is a set of clauses φ'' that are required to be added to φ' so that the resulting MAXSAT formula $\varphi' \cup \varphi''$ is solved in the next iteration of Algorithm 1. If φ'' is empty, then this means that Algorithm 1 can stop and return the appropriate model (as described in line 8 of Algorithm 1).

CHECK starts by calling the function APPROX (line 3) which takes φ, φ' and $\alpha_{\varphi'}$ as inputs, and returns a new set of clauses ψ, which we refer to as the *summarization set*. APPROX analyzes clauses in $\varphi \setminus \varphi'$, and returns a much smaller formula ψ which allows us to overestimate the gap between the optimum objective value under current partial model $\alpha_{\varphi'}$ and the optimum objective value of φ. In line 5, CHECK computes a model $\alpha_{\varphi''}$ of the MAXSAT formula $\varphi'' = \varphi' \cup \psi$. Next, in line 6, CHECK compares the objective value of φ' with respect to $\alpha_{\varphi'}$ and the objective value of φ'' with respect to $\alpha_{\varphi''}$. If these objective values are equal, CHECK concludes that the partial assignment for the queries in $\alpha_{\varphi'}$ is a model to the Q-MAXSAT problem and returns an empty set (line 7). Otherwise, it computes φ_s in line 9 which is the set of clauses satisfied by $\alpha_{\varphi''}$ in ψ. Finally, in line 10, CHECK returns the set of clauses to be added to the MAXSAT formula φ'. This is computed by REFINE, which takes φ_s, φ, φ', and $\alpha_{\varphi'}$ as input. Essentially, REFINE identifies the clauses in $\varphi \setminus \varphi'$ which are likely responsible for failing the check in line 6, and uses them to expand the MAXSAT formula φ'.

***Optimality check via* APPROX.** The core step of CHECK is line 6, which uses $eval(\varphi'', \alpha_{\varphi''}) - eval(\varphi', \alpha_{\varphi'})$ to overestimate the gap between the optimum objective value under current partial model $\alpha_{\varphi'}$ and the optimum objective value of φ. The key idea here is to apply APPROX to generate a *small* set of clauses ψ which succinctly summarizes the effect of the unexplored clauses $\varphi \setminus \varphi'$. We next describe the specification of the APPROX function, and formally prove the soundness of the optimality check in line 6.

Given a set of variables $\mathcal{U} \subseteq Var(\varphi)$, and an assignment $\alpha_{\mathcal{U}}$, we define a *substitution operation* $\varphi[\alpha_{\mathcal{U}}]$ which simplifies φ by replacing all occurrences of variables in $\mathcal{U}$ with their corresponding values in assignment $\alpha_{\mathcal{U}}$. Formally,

$$\begin{aligned}
\{s_1, ..., s_n\}[\alpha_{\mathcal{U}}] &= \{s_1[\alpha_{\mathcal{U}}], ..., s_n[\alpha_{\mathcal{U}}]\} \\
(w, c)[\alpha_{\mathcal{U}}] &= (w, c[\alpha_{\mathcal{U}}]) \\
1[\alpha_{\mathcal{U}}] &= 1 \\
0[\alpha_{\mathcal{U}}] &= 0
\end{aligned}$$

$$(\textstyle\bigvee_{i=1}^{n} v_i \lor \bigvee_{j=1}^{m} \neg v'_j)[\alpha_{\mathcal{U}}] = \begin{cases} 1, & \text{if } \alpha_{\mathcal{U}} \models \bigvee_{i=1}^{n} v_i \lor \bigvee_{j=1}^{m} \neg v'_j, \\ 0, & \text{if } \{v_1, .., v_n\} \subseteq \mathcal{U} \land \{v'_1, ..., v'_m\} \subseteq \mathcal{U} \land \\ & \quad \alpha_{\mathcal{U}} \not\models \bigvee_{i=1}^{n} v_i \lor \bigvee_{j=1}^{m} \neg v'_j, \\ \bigvee_{v \in \{v_1,...,v_n\} \setminus \mathcal{U}} v \lor \bigvee_{v' \in \{v'_1,...,v'_n\} \setminus \mathcal{U}} \neg v', & \text{otherwise.} \end{cases}$$

Definition 2 (Summarizing unexplored clauses). Given two MAXSAT formulae φ and φ' such that $\varphi' \subseteq \varphi$, and $\alpha_{\varphi'} \in MaxSAT(\varphi')$, we say $\psi = \text{APPROX}(\varphi, \varphi', \alpha_{\varphi'})$ *summarizes* the effect of $\varphi \setminus \varphi'$ with respect to $\alpha_{\varphi'}$ if and only if:

$$\max_\alpha eval(\varphi' \cup \psi, \alpha) \geq \max_\alpha eval(\varphi, \alpha) - \max_\alpha eval((\varphi \setminus \varphi')[\alpha_{\varphi'}], \alpha)$$

We state two useful facts about $eval$ before proving soundness.

Proposition 3. Let φ and φ' be two MAXSAT formulae such that $\varphi \cap \varphi' = \emptyset$. Then, $\forall \alpha . eval(\varphi \cup \varphi', \alpha) = eval(\varphi, \alpha) + eval(\varphi', \alpha)$.

Proposition 4. Let φ and φ' be two MAXSAT formulae. Then,

$$\max_\alpha \ eval(\varphi, \alpha) + \max_\alpha \ eval(\varphi', \alpha) \geq \max_\alpha \ (eval(\varphi, \alpha) + eval(\varphi', \alpha)).$$

The theorem below states the soundness of the optimality check performed in line 6 in CHECK.

Theorem 5 (Soundness of optimality check). *Given a* Q-MAXSAT *instance* $(\varphi, \mathcal{Q})$*, a* MAXSAT *formula* $\varphi' \subseteq \varphi$ *s.t.* $Vars(\varphi') \supseteq \mathcal{Q}$*, a model* $\alpha_{\varphi'} \in MaxSAT(\varphi')$*, and* $\psi = \text{APPROX}(\varphi, \varphi', \alpha_{\varphi'})$ *such that the following condition holds:*

$$\max_\alpha eval(\varphi' \cup \psi, \alpha) = \max_\alpha eval(\varphi', \alpha).$$

Then, $\lambda v . \alpha_{\varphi'}(v)$*, where* $v \in \mathcal{Q}$ *is a model of the* Q-MAXSAT *instance* $(\varphi, \mathcal{Q})$.

Proof. Let $\alpha_{\mathcal{Q}} = \lambda v . \alpha_{\varphi'}(v)$, where $v \in \mathcal{Q}$. Let $\varphi'' = \varphi[\alpha_{\varphi'}]$. Construct a completion α_φ of $\alpha_{\mathcal{Q}}$ as follows:

$$\alpha_\varphi = \alpha_{\varphi'} \uplus \alpha_{\varphi''}, \text{ where } \alpha_{\varphi''} \in MaxSAT(\varphi'')$$

It suffices to show that $\alpha_\varphi \in MaxSAT(\varphi)$, that is, $eval(\varphi, \alpha_\varphi) = \max_\alpha \ eval(\varphi, \alpha)$. We have:

$$\begin{aligned}
& eval(\varphi, \alpha_\varphi) \\
&= eval(\varphi, \alpha_{\varphi'} \uplus \alpha_{\varphi''}) \\
&= eval(\varphi[\alpha_{\varphi'}], \alpha_{\varphi''}) \\
&= \max_\alpha eval(\varphi[\alpha_{\varphi'}], \alpha) \\
&\qquad [\text{since } \varphi'' = \varphi[\alpha_{\varphi'}] \text{ and } \alpha_{\varphi''} \in MaxSAT(\varphi'')] \\
&= \max_\alpha eval(\varphi'[\alpha_{\varphi'}] \cup (\varphi \setminus \varphi')[\alpha_{\varphi'}], \alpha) \\
&= \max_\alpha (eval(\varphi'[\alpha_{\varphi'}], \alpha) + eval((\varphi \setminus \varphi')[\alpha_{\varphi'}], \alpha)) \quad [\text{from Prop. 3}] \\
&= \max_\alpha (eval(\varphi', \alpha_{\varphi'}) + eval((\varphi \setminus \varphi')[\alpha_{\varphi'}], \alpha)) \\
&\qquad [\text{since } \alpha_{\varphi'} \in MaxSAT(\varphi')]
\end{aligned}$$

$$= eval(\varphi', \alpha_{\varphi'}) + \max_{\alpha} eval((\varphi \setminus \varphi')[\alpha_{\varphi'}], \alpha)$$
$$\geq eval(\varphi', \alpha_{\varphi'}) + \max_{\alpha} eval(\varphi, \alpha) - \max_{\alpha} eval(\varphi' \cup \psi, \alpha)$$
[since $\psi = \text{APPROX}(\varphi, \varphi', \alpha_{\varphi'})$, see Defn. 2]
$$= \max_{\alpha} eval(\varphi', \alpha) + \max_{\alpha} eval(\varphi, \alpha) - \max_{\alpha} eval(\varphi' \cup \psi, \alpha)$$
[since $\alpha_{\varphi'} \in MaxSAT(\varphi')$]
$$= \max_{\alpha} eval(\varphi' \cup \psi, \alpha) + \max_{\alpha} eval(\varphi, \alpha) - \max_{\alpha} eval(\varphi' \cup \psi, \alpha)$$
[from the condition defined in the theorem statement]
$$= \max_{\alpha} eval(\varphi, \alpha) \qquad \square$$

Discussion. There are number of possibilities for the APPROX function that satisfy the specification in Definition 2. The quality of such an APPROX function can be measured by two criteria: the efficiency and the precision of the optimality check (lines 3 and 6 in Algorithm 2).

Given that $eval$ can be efficiently computed, the cost of the optimality check primarily depends on the cost of computing ψ via APPROX, and the cost of invoking the MAXSAT solver on $\varphi' \cup \psi$. Since MAXSAT is known to be a computationally hard problem, a simple ψ returned by APPROX can significantly speedup the optimality check.

On the other hand, a precise optimality check can significantly reduce the number of iterations of Algorithm 1. We define a partial order $\preceq$ on APPROX functions that compares the precision of the optimality check via them. We say APPROX_1 is more precise than APPROX_2 (denoted by $\text{APPROX}_2 \preceq \text{APPROX}_1$), if for any given MAXSAT formulae φ, $\varphi' \subseteq \varphi$, and $\alpha_{\varphi'} \in MaxSAT(\varphi')$, the optimum objective value of $\varphi' \cup \text{APPROX}_1(\varphi, \varphi', \alpha_{\varphi'})$ is no larger than that of $\varphi' \cup \text{APPROX}_2(\varphi, \varphi', \alpha_{\varphi'})$. More formally:

$$\text{APPROX}_2 \preceq \text{APPROX}_1 \Leftrightarrow \forall \varphi, \varphi' \subseteq \varphi, \alpha_{\varphi'} \in MaxSAT(\varphi'):$$
$$\max_\alpha eval(\varphi' \cup \psi_1, \alpha) \leq \max_\alpha eval(\varphi' \cup \psi_2, \alpha),$$
$$\text{where } \psi_1 = \text{APPROX}_1(\varphi, \varphi', \alpha_{\varphi'}), \psi_2 = \text{APPROX}_2(\varphi, \varphi', \alpha_{\varphi'}).$$

In Section 4.2 we introduce three different APPROX functions with increasing order of precision. While the more precise APPROX operators reduce the number of iterations in our algorithm, they are more expensive to compute.

***Expanding relevant clauses via* REFINE.** The REFINE function finds a new set of clauses that are relevant to the queries, and adds them to the workset when the optimality check fails. To guarantee the termination of our approach, when the optimality check fails, REFINE should always return a nonempty set. We describe the details of various REFINE functions in Section 4.2.

4.2 Efficient Optimality Check via Distinct APPROX Functions

We introduce three different APPROX functions and their corresponding REFINE functions to construct efficient CHECK functions. These three functions are the ID-APPROX function, the π-APPROX function, and the γ-APPROX function. Each function is constructed by extending the previous one, and their precision order is:

$$\text{ID-APPROX} \preceq \pi\text{-APPROX} \preceq \gamma\text{-APPROX}.$$

The cost of executing each of these APPROX functions also increases with precision. After defining each function and proving that it satisfies the specification of an APPROX function, we discuss the efficiency and the precision of the optimality check using it.

4.2.1 The ID-APPROX Function

The ID-APPROX function is based on the following observation: for a Q-MAXSAT instance, the clauses not explored by Algorithm 1 can only affect the assignments to the queries via the clauses sharing variables with the workset. We refer to such unexplored clauses as *frontier clauses*. If all the frontier clauses are satisfied by the current partial model, or they cannot further improve the objective value of the workset, we can construct a model of the Q-MAXSAT instance from the current partial model. Based on these observations, the ID-APPROX function constructs the summarization set by adding all the frontier clauses that are not satisfied by the current partial model.

To define ID-APPROX, we first define what it means to say that a clause is satisfied by a partial model. A clause is satisfied by a partial model over a set of variables $\mathcal{U}$, if it is satisfied by all completions under that partial model. In other words:

$$\alpha_{\mathcal{U}} \models \bigvee_{i=1}^{n} v_i \vee \bigvee_{j=1}^{m} \neg v'_j \Leftrightarrow (\exists i : (v_i \in \mathcal{U}) \wedge \alpha_{\mathcal{U}}(v_i) = 1)$$
$$\text{or } (\exists j : (v'_j \in \mathcal{U}) \wedge \alpha_{\mathcal{U}}(v'_j) = 0).$$

Definition 6 (ID-APPROX)**.** Given formulae φ, $\varphi' \subseteq \varphi$, and a partial model $\alpha_{\varphi'} \in MaxSAT(\varphi')$, ID-APPROX is defined as follows:

$$\text{ID-APPROX}(\varphi, \varphi', \alpha_{\varphi'}) = \{(w, c) \mid (w, c) \in (\varphi \setminus \varphi') \wedge$$
$$Var(\{(w, c)\}) \cap Var(\varphi') \neq \emptyset \wedge \alpha_{\varphi'} \not\models c)\}.$$

The corresponding REFINE function is:

$$\text{ID-REFINE}(\varphi_s, \varphi, \varphi', \alpha_{\varphi'}) = \varphi_s.$$

Example. Let $(\varphi, \mathcal{Q})$ be a Q-MAXSAT instance, where $\varphi = \{(2, x), (5, \neg x \vee y), (5, y \vee z), (1, \neg y)\}$ and $\mathcal{Q} = \{x, y\}$. Assume that the workset is $\varphi' = \{(2, x), (5, \neg x \vee y)\}$. By invoking a MAXSAT solver on φ', we get a model $\alpha_{\mathcal{Q}} = [x \mapsto 1, y \mapsto 1]$. Both clauses in $\varphi \setminus \varphi'$ contain y, where $\varphi \setminus \varphi' = \{(5, y \vee z), (1, \neg y)\}$. Since $(1, \neg y)$ is not satisfied by $\alpha_{\mathcal{Q}}$, ID-APPROX$(\varphi, \varphi', \alpha_{\varphi'})$ produces $\psi = \{(1, \neg y)\}$. As the optimum objective values of both φ' and $\varphi' \cup \psi$ are 7, we conclude $[x \mapsto 1, y \mapsto 1]$ is a model of the given Q-MAXSAT instance. Indeed, its completion $[x \mapsto 1, y \mapsto 1, z \mapsto 0]$ is a model of the MAXSAT formula φ. $\square$

Theorem 7 (Soundness of ID-APPROX**).** *ID-APPROX$(\varphi, \varphi', \alpha_{\varphi'})$ summarizes the effect of $\varphi \setminus \varphi'$ with respect to $\alpha_{\varphi'}$.*

Proof. Let ψ=ID-APPROX$(\varphi, \varphi', \alpha_{\varphi'})$. We show that ID-APPROX satisfies the specification of an APPROX function in Definition 2 by proving the inequality below:

$$\max_{\alpha}\ eval(\varphi' \cup \psi, \alpha) + \max_{\alpha}\ eval(\varphi \setminus \varphi'[\alpha_{\varphi'}], \alpha) \geq \max_{\alpha}\ eval(\varphi, \alpha)$$

We use φ_1 to represent the set of frontier clauses, and φ_2 to represent the rest of the clauses in $\varphi \setminus \varphi'$:

$$\varphi_1 = \{(w, c) \in (\varphi \setminus \varphi') \mid Var(\{(w, c)\}) \cap Var(\varphi') \neq \emptyset\}$$
$$\varphi_2 = \{(w, c) \in (\varphi \setminus \varphi') \mid Var(\{(w, c)\}) \cap Var(\varphi') = \emptyset\}$$

We further split the set of frontier clauses φ_1 into two sets:

$$\varphi'_1 = \{(w, c) \in \varphi_1 \mid \alpha_{\varphi'} \not\models c\}, \quad \varphi''_1 = \{(w, c) \in \varphi_1 \mid \alpha_{\varphi'} \models c\}$$

φ'_1 is effectively what is returned by ID-APPROX$(\varphi, \varphi', \alpha_{\varphi'})$. We first prove the following claim:

$$\forall \alpha. eval(\varphi''_1[\alpha_{\varphi'}] \cup \varphi_2, \alpha) \geq eval(\varphi''_1 \cup \varphi_2, \alpha) \qquad (1)$$

We have:

$eval(\varphi''_1[\alpha_{\varphi'}] \cup \varphi_2, \alpha)$

$= eval(\varphi''_1[\alpha_{\varphi'}], \alpha) + eval(\varphi_2, \alpha)$ [from Prop. 3]

$= \max_{\alpha} eval(\varphi''_1, \alpha) + eval(\varphi_2, \alpha)$ [since $\forall (w, c) \in \varphi''_1 . \alpha_{\varphi'} \models c$]

$\geq eval(\varphi''_1, \alpha) + eval(\varphi_2, \alpha)$

$= eval(\varphi''_1 \cup \varphi_2, \alpha)$ [from Prop. 3]

Now we prove the theorem. We have:

$$\begin{aligned}
&\max_\alpha\ eval(\varphi' \cup \psi, \alpha) + \max_\alpha\ eval(\varphi \setminus \varphi'[\alpha_{\varphi'}], \alpha) \\
&= \max_\alpha eval(\varphi' \cup \varphi'_1, \alpha) + \max_\alpha eval(\varphi \setminus \varphi'[\alpha_{\varphi'}], \alpha) \\
&\qquad [\text{since } \psi = \text{ID-APPROX}(\varphi, \varphi', \alpha_{\varphi'}), \text{see Defn. 6}] \\
&= \max_\alpha eval(\varphi' \cup \varphi'_1, \alpha) + \\
&\quad \max_\alpha eval(\varphi'_1[\alpha_{\varphi'}] \cup \varphi''_1[\alpha_{\varphi'}] \cup \varphi_2[\alpha_{\varphi'}], \alpha) \\
&\geq \max_\alpha eval(\varphi' \cup \varphi'_1, \alpha) + \max_\alpha eval(\varphi''_1[\alpha_{\varphi'}] \cup \varphi_2[\alpha_{\varphi'}], \alpha) \\
&\qquad [\text{since } \forall \varphi, \varphi'. \max_\alpha eval(\varphi \cup \varphi') \geq \max_\alpha eval(\varphi)] \\
&= \max_\alpha eval(\varphi' \cup \varphi'_1, \alpha) + \max_\alpha eval(\varphi''_1[\alpha_{\varphi'}] \cup \varphi_2, \alpha) \\
&\qquad [\text{since } Var(\varphi') \cap Var(\varphi_2) = \emptyset] \\
&\geq \max_\alpha eval(\varphi' \cup \varphi'_1, \alpha) + \max_\alpha eval(\varphi''_1 \cup \varphi_2, \alpha) \quad [\text{from (1)}] \\
&\geq \max_\alpha (eval(\varphi' \cup \varphi'_1, \alpha) + eval(\varphi''_1 \cup \varphi_2, \alpha)) \quad [\text{from Prop. 4}] \\
&= \max_\alpha\ eval(\varphi' \cup \varphi'_1 \cup \varphi''_1 \cup \varphi_2, \alpha) \quad [\text{from Prop. 3}] \\
&= \max_\alpha\ eval(\varphi, \alpha) \quad \square
\end{aligned}$$

Discussion. The ID-APPROX function effectively exploits the observation that, in a potentially very large Q-MAXSAT instance, the unexplored part of the formula can only impact the assignments to the queries via the frontier clauses. In practice, the set of frontier clauses is usually much smaller compared to the set of all unexplored clauses, resulting in an efficient invocation of the MAXSAT solver in the optimality check. Moreover, ID-APPROX is cheap to compute as it can be implemented via a linear scan of the unexplored clauses. However, the precision of ID-APPROX may not be satisfactory, causing the optimality check to be overly conservative. One such scenario is that, if the clauses in the summarization set generated from ID-APPROX contain any variable that is not used in any clause in the workset, then the check will fail. To overcome this limitation, we next introduce the π-APPROX function.

4.2.2 The π-APPROX Function

π-APPROX improves the precision of ID-APPROX by exploiting the following observation: if the frontier clauses violated by the current partial model have relatively low weights, even though they may contain new variables, it is very likely the case that we can resolve the queries with the workset. To overcome the limitation of ID-APPROX, π-APPROX generates the summarization set by applying a strengthening function on the frontier clauses violated by the current partial model.

We define the strengthening function *retain* below.

Definition 8 (*retain*). We define *retain*$(c, \mathcal{V})$ as follows:

$$\begin{aligned}
retain(1, \mathcal{V}) &= 1 \\
retain(0, \mathcal{V}) &= 0 \\
retain(\textstyle\bigvee_{i=1}^n v_i \vee \bigvee_{j=1}^m \neg v'_j, \mathcal{V}) &= \\
\text{let } \mathcal{V}_1 = \mathcal{V} \cap \{v_1, .., v_n\} &\text{ and } \mathcal{V}_2 = \mathcal{V} \cap \{v'_1, ..., v'_m\} \\
\text{in } &\begin{pmatrix} 0 & \text{if } \mathcal{V}_1 = \mathcal{V}_2 = \emptyset \\ \bigvee_{u \in \mathcal{V}_1} u \vee \bigvee_{u' \in \mathcal{V}_2} \neg u' & \text{otherwise} \end{pmatrix}
\end{aligned}$$

Definition 9 (π-APPROX). Given a formula φ, a formula $\varphi' \subseteq \varphi$, and $\alpha_{\varphi'} \in MaxSAT(\varphi')$, π-APPROX is defined as follows:

$$\begin{aligned}
&\pi\text{-APPROX}(\varphi, \varphi', \alpha_{\varphi'}) = \{\ (w, retain(c,\ Var(\varphi')))\ | \\
&(w, c) \in \varphi \setminus \varphi' \wedge\ Var(\{(w, c)\}) \cap Var(\varphi') \neq \emptyset \wedge \alpha_{\varphi'} \not\models c)\ \}
\end{aligned}$$

The corresponding REFINE function is:

$$\begin{aligned}
\pi\text{-REFINE}(\varphi_s, \varphi, \varphi', \alpha_{\varphi'}) = \{(w, c) \in \varphi \setminus \varphi'\ | \\
(w, retain(c,\ Var(\varphi'))) \in \varphi_s\}
\end{aligned}$$

Example. Let $(\varphi, \mathcal{Q})$ where $\varphi = \{(5, x), (5, \neg x \vee y), (1, \neg y \vee z), (5, \neg z)\}$ and $Q = \{x, y\}$ be a Q-MAXSAT instance. Assume that the workset is $\varphi' = \{(5, x), (5, \neg x \vee y)\}$. By invoking a MAXSAT solver on φ', we get $\alpha_{\mathcal{Q}} = [x \mapsto 1, y \mapsto 1]$ with the objective value of φ' being 10. The only clause in $\varphi \setminus \varphi'$ that uses x or y is $(1, \neg y \vee z)$, which is not satisfied by $\alpha_{\mathcal{Q}}$. By applying π-APPROX, we generate the summarization set $\psi = \{(1, \neg y)\}$. By invoking a MAXSAT solver on $\varphi' \cup \psi$, we find its optimum objective value to be 10, which is the same as that of φ'. Thus, we conclude that $\alpha_{\mathcal{Q}}$ is a solution of Q-MAXSAT instance $(\varphi, \{x, y\})$. Indeed, model $[x \mapsto 1, y \mapsto 1, z \mapsto 0]$ which is a completion of α_Q, is a solution of the MAXSAT formula φ. On the other hand, the optimality check using the ID-APPROX function will return $\psi' = \{(1, \neg y \vee z)\}$ as the summarization set. By invoking the MAXSAT solver on $\phi' \cup \psi'$, we get an optimum objective value of 11 with $[x \mapsto 1, y \mapsto 1, z \mapsto 1]$. As a result, the optimality check with ID-APPROX fails here because of the presence of z in the summarization set. $\square$

To prove that π-APPROX satisfies the specification of APPROX in Definition 2, we first introduce a decomposition function.

Definition 10 (π-DECOMP). Given a formula φ and set of variables $\mathcal{V} \subseteq Var(\varphi)$, let $\mathcal{V}' = Var(\varphi) \setminus \mathcal{V}$. Then, we define π-DECOMP$(\varphi, \mathcal{V}) = (\varphi_1, \varphi_2)$ such that:

$$\begin{aligned}
\varphi_1 &= \{\ (w, retain(c, \mathcal{V}))\ |\ (w, c) \in \varphi \wedge Var(\{(w, c)\}) \cap \mathcal{V} \neq \emptyset\ \} \\
\varphi_2 &= \{\ (w, retain(c, \mathcal{V}'))\ |\ (w, c) \in \varphi \wedge \\
&\qquad (Var(\{(w, c)\}) \cap \mathcal{V}' \neq \emptyset \vee c = 1 \vee c = 0)\ \}.
\end{aligned}$$

Lemma 11. *Let π-DECOMP$(\varphi, \mathcal{V}) = (\varphi_1, \varphi_2)$, where $\mathcal{V} \subseteq Var(\varphi)$. We have $eval(\varphi_1 \cup \varphi_2, \alpha) \geq eval(\varphi, \alpha)$ for all α.*

Proof. To simplify the proof, we first remove the clauses with no variables from both φ and $\varphi_1 \cup \varphi_2$. We use the following two sets to represent such clauses:

$$\varphi_3 = \{(w, c) \in \varphi \mid c = 1\}, \quad \varphi_4 = \{(w, c) \in \varphi \mid c = 0\}.$$

We also define the following two sets:

$$\varphi'_3 = \{(w, c) \in \varphi_2 \mid c = 1\}, \quad \varphi'_4 = \{(w, c) \in \varphi_2 \mid c = 0\}.$$

Based on the definition of φ_2, we have $\varphi_3 = \varphi'_3$ and $\varphi_4 = \varphi'_4$. Therefore, the inequality in the lemma can be rewritten as:

$$eval(\varphi_1 \cup \varphi_2 \setminus (\varphi'_3 \cup \varphi'_4), \alpha) \geq eval(\varphi \setminus (\varphi_3 \cup \varphi_4), \alpha).$$

From this point on, we assume $\varphi = \varphi \setminus (\varphi_3 \cup \varphi_4)$ and $\varphi_2 = \varphi_2 \setminus (\varphi'_3 \cup \varphi'_4)$. Let $\mathcal{V}' = Var(\varphi) \setminus \mathcal{V}$. We prove the lemma by showing that for any $s \in \varphi$, where $s = (w, c)$ and α,

$$\begin{aligned}
eval(\{(w, retain(c, \mathcal{V}))\}, \alpha) + eval(\{(w, retain(c, \mathcal{V}'))\}, \alpha) \\
\geq eval(\{(w, c\}, \alpha) \qquad (1)
\end{aligned}$$

Let $S = \{(w, c)\}$, $S_1 = \{(w, retain(c, \mathcal{V}))\}$, $S_2 = \{(w, retain(c, \mathcal{V}'))\}$. We prove the above claim with respect to the three different cases:

1. If $Var(S) \cap \mathcal{V} = \emptyset$, then we have $S_1 = \{(w, 0)\}$, $S_2 = S$. Since $\forall w, \alpha\ .\ eval(\{w, 0\}, \alpha) = 0$, we have $\forall \alpha\ .\ eval(S_1, \alpha) + eval(S_2, \alpha) = eval(S, \alpha)$.
2. If $Var(S) \cap \mathcal{V}' = \emptyset$, then we have $S_1 = S$, $S_2 = \{(w, 0)\}$. Similar to Case 1, we have $\forall \alpha\ .\ eval(S_1, \alpha) + eval(S_2, \alpha) = eval(S, \alpha)$.
3. If $Var(S) \cap \mathcal{V} \neq \emptyset$ and $Var(S) \cap \mathcal{V}' \neq \emptyset$, we prove the case by converting S, S_1 and S_2 into their equivalent pseudo-Boolean functions. A pseudo-Boolean function f is a multi-linear polynomial, which maps the assignment of a set of Boolean variables to a real number:

$$f(v_1, ..., v_n)\ =\ a_0 + \sum_{i=1}^{m} (a_i \prod_{j=1}^{p} v_j), \quad \text{where } a_i \in \mathbb{R}^+.$$

Any MaxSat formula can be converted into a pseudo-Boolean function. The conversion $\langle\cdot\rangle$ is as follows:

$$\begin{aligned}\langle\{s_1,...,s_n\}\rangle &= \textstyle\sum_{i=1}^n(fst(s_i)-fst(s_i)\langle snd(s_i)\rangle)\\ \langle\textstyle\bigvee_{i=1}^n v_i \vee \bigvee_{j=1}^m \neg v'_j\rangle &= \textstyle\prod_{i=1}^n(1-v_i) \,*\, \prod_{j=1}^m v'_j\\ \langle 1\rangle &= 0\\ \langle 0\rangle &= 1\end{aligned}$$

For example, the MaxSat formula $\{(5,\neg w\vee x),(3,y\vee z)\}$ is converted into function $5-5w(1-x)+3-3(1-y)(1-z)$. We use $eval(f,\alpha)$ to denote the evaluation of pseudo-Boolean function f under model α. From the conversion, we can conclude that $\forall\varphi,\alpha\,.\,eval(\varphi,\alpha)=eval(\langle\varphi\rangle,\alpha)$.

We now prove the claim (1) under the third case above. We rewrite $c=\bigvee_{i=1}^n v_i\vee\bigvee_{j=1}^m\neg v'_j$ as below:

$$c=\textstyle\bigvee_{x\in\mathcal{V}_1}x\vee\bigvee_{x'\in\mathcal{V}_2}\neg x'\vee\bigvee_{y\in\mathcal{V}_3}y\vee\bigvee_{y'\in\mathcal{V}_4}\neg y'$$

where $\mathcal{V}_1=\mathcal{V}\cap\{v_1,..,v_n\}$, $\mathcal{V}_2=\mathcal{V}\cap\{v'_1,..,v'_m\}$, $\mathcal{V}_3=\mathcal{V}'\cap\{v_1,..,v_n\}$ and $\mathcal{V}_4=\mathcal{V}'\cap\{v'_1,..,v'_m\}$.

Let $c_1=\bigvee_{x\in\mathcal{V}_1}x\vee\bigvee_{x'\in\mathcal{V}_2}\neg x'$ and $c_2=\bigvee_{y\in\mathcal{V}_3}y\vee\bigvee_{y'\in\mathcal{V}_4}\neg y'$. Then, we have $S_1=\{(w,c_1)\}$ and $S_2=\{(w,c_2)\}$. It suffices to prove that $\langle S_1\rangle+\langle S_2\rangle-\langle S\rangle\geq 0$. We have:

$$\begin{aligned}&\langle S_1\rangle+\langle S_2\rangle-\langle S\rangle\\ &=w-w\prod_{x\in\mathcal{V}_1}(1-x)\prod_{x'\in\mathcal{V}_2}x'+w-w\prod_{y\in\mathcal{V}_3}(1-y)\prod_{y'\in\mathcal{V}_4}y'\\ &\quad-(w-w\prod_{x\in\mathcal{V}_1}(1-x)\prod_{x'\in\mathcal{V}_2}x'\prod_{y\in\mathcal{V}_3}(1-y)\prod_{y'\in\mathcal{V}_4}y')\\ &=w(1-\prod_{x\in\mathcal{V}_1}(1-x)\prod_{x'\in\mathcal{V}_2}x')-\\ &\quad w\prod_{y\in\mathcal{V}_3}(1-y)\prod_{y'\in\mathcal{V}_4}y'(1-\prod_{x\in\mathcal{V}_1}(1-x)\prod_{x'\in\mathcal{V}_2}x')\\ &=w(1-\prod_{x\in\mathcal{V}_1}(1-x)\prod_{x'\in\mathcal{V}_2}x')(1-\prod_{y\in\mathcal{V}_3}(1-y)\prod_{y'\in\mathcal{V}_4}y')\end{aligned}$$

Since all variables are Boolean variables, we have

$$\begin{aligned}1-\textstyle\prod_{x\in\mathcal{V}_1}(1-x)*\prod_{x'\in\mathcal{V}_2}x'\geq 0 \text{ and}\\ 1-\textstyle\prod_{y\in\mathcal{V}_3}(1-y)*\prod_{y'\in\mathcal{V}_4}y'\geq 0.\end{aligned}$$

□

From Lemma 11, Propositions 3 and 4, we can also conclude that

$$\begin{aligned}&\max_\alpha eval(\varphi_1,\alpha)+\max_\alpha eval(\varphi_2,\alpha)\\ &\geq\max_\alpha eval(\varphi_1\cup\varphi_2,\alpha)\geq\max_\alpha eval(\varphi,\alpha)\end{aligned}$$

where π-Decomp$(\varphi,\mathcal{V})=(\varphi_1,\varphi_2)$.

We now use the lemma to prove that π-Approx is a approx function as defined in Definition 2.

Theorem 12 (Soundness of π-Approx). *π-Approx$(\varphi,\varphi',\alpha_{\varphi'})$ summarizes the effect of $\varphi\setminus\varphi'$ with respect to $\alpha_{\varphi'}$.*

Proof. Let $\psi=\pi$-Approx$(\varphi,\varphi',\alpha_{\varphi'})$. We show that π-Approx satisfies the specification of an approx function in Definition 2 by proving the inequality below:

$$\max_\alpha eval(\varphi'\cup\psi,\alpha)+\max_\alpha eval(\varphi\setminus\varphi'[\alpha_{\varphi'}],\alpha)\geq\max_\alpha eval(\varphi,\alpha)$$

Without loss of generality, we assume that φ does not contain any clause (w,c) where $c=0$ or $c=1$. Otherwise, we can rewrite the inequality above by removing these clauses from φ, φ', and $\varphi\setminus\varphi'$.

As in the soundness proof for ID-Approx, we define φ_1,φ_2, φ'_1, and φ''_1 as follows:

$$\begin{aligned}\varphi_1&=\{(w,c)\in(\varphi\setminus\varphi')\mid Var(\{(w,c)\})\cap Var(\varphi')\neq\emptyset\},\\ \varphi_2&=\{(w,c)\in(\varphi\setminus\varphi')\mid Var(\{(w,c)\})\cap Var(\varphi')=\emptyset\},\\ \varphi'_1&=\{(w,c)\in\varphi_1\mid\alpha_{\varphi'}\not\models c\},\\ \varphi''_1&=\{(w,c)\in\varphi_1\mid\alpha_{\varphi'}\models c\}.\end{aligned}$$

We have:

$$\begin{aligned}&\max_\alpha eval(\varphi'\cup\psi,\alpha)+\max_\alpha eval((\varphi\setminus\varphi')[\alpha_{\varphi'}],\alpha)\\ &=\max_\alpha eval(\varphi'\cup\psi,\alpha)+\\ &\quad\max_\alpha eval(\varphi'_1[\alpha_{\varphi'}]\cup\varphi''_1[\alpha_{\varphi'}]\cup\varphi_2[\alpha_{\varphi'}],\alpha)\\ &=\max_\alpha eval(\varphi'\cup\psi,\alpha)+\max_\alpha(eval(\varphi''_1[\alpha_{\varphi'}],\alpha)+\\ &\quad eval(\varphi'_1[\alpha_{\varphi'}]\cup\varphi_2[\alpha_{\varphi'}],\alpha))\quad\text{[from Prop. 3]}\\ &=\max_\alpha eval(\varphi'\cup\psi,\alpha)+\max_\alpha eval(\varphi''_1,\alpha)+\\ &\quad\max_\alpha eval(\varphi'_1[\alpha_{\varphi'}]\cup\varphi_2[\alpha_{\varphi'}],\alpha)\ \ \text{[since }\forall(w,c)\in\varphi''_1.\alpha_{\varphi'}\models c]\\ &=\max_\alpha eval(\varphi'\cup\psi,\alpha)+\max_\alpha eval(\varphi''_1,\alpha)+\\ &\quad\max_\alpha eval(\varphi'_1[\alpha_{\varphi'}]\cup\varphi_2,\alpha)\ \ \text{[since }Var(\varphi')\cap Var(\varphi_2)=\emptyset]\end{aligned}$$

Next, we show that π-Decomp$(\varphi'\cup\varphi'_1\cup\varphi_2,Var(\varphi'))=(\varphi'\cup\psi,\varphi'_1[\alpha_{\varphi'}]\cup\varphi_2)$. Let $\mathcal{V}_1=Var(\varphi')$ and $\mathcal{V}_2=Var(\varphi)\setminus\mathcal{V}_1$. For a given clause $(w,c)\in\varphi'\cup\varphi'_1\cup\varphi_2$, we show the result of $(w,retain(c,\mathcal{V}_1))$ and $(w,retain(c,\mathcal{V}_2))$ under the following cases:

1. When $(w,c)\in\varphi'$, $retain(c,\mathcal{V})=c$ and $retain(c,\mathcal{V}')=0$. Hence we have $\varphi'=\{(w,retain(c,\mathcal{V}))\mid(w,c)\in\varphi'\wedge retain(c,\mathcal{V})\neq 0\}$
2. When $(w,c)\in\varphi_2$, $retain(c,\mathcal{V})=0$ and $retain(c,\mathcal{V}')=c$. Hence we have $\varphi_2=\{(w,retain(c,\mathcal{V}'))\mid(w,c)\in\varphi_2\wedge retain(c,\mathcal{V}')\neq 0\}$
3. When $(w,c)\in\varphi'_1$, based on the definition of ψ and φ'_1, we have $\psi=\{(w,retain(c,\mathcal{V}))\mid(w,c)\in\varphi'_1\wedge retain(c,\mathcal{V})\neq 0\}$ and $\varphi'_1[\alpha_{\varphi'}]=\{(w,retain(c,\mathcal{V}'))\mid(w,c)\in\varphi'_1\wedge retain(c,\mathcal{V}')\neq 0\}$.

Therefore, π-Decomp$(\varphi'\cup\varphi'_1\cup\varphi_2,Var(\varphi'))=(\varphi'\cup\psi,\varphi'_1[\alpha_{\varphi'}]\cup\varphi_2)$.

By applying Lemma 11, we can derive

$$\begin{aligned}&\max_\alpha eval(\varphi'\cup\psi,\alpha)+\max_\alpha eval(\varphi_1[\alpha_{\varphi'}]\cup\varphi_2,\alpha)\\ &\qquad\geq\max_\alpha eval(\varphi'\cup\varphi'_1\cup\varphi_2,\alpha).\end{aligned}\qquad(1)$$

Thus, we can prove the theorem as follows:

$$\begin{aligned}&\max_\alpha eval(\varphi'\cup\psi,\alpha)+\max_\alpha eval(\varphi\setminus\varphi'[\alpha_{\varphi'}],\alpha)\\ &=\max_\alpha eval(\varphi'\cup\psi,\alpha)+\max_\alpha eval(\varphi''_1,\alpha)+\\ &\quad\max_\alpha eval(\varphi'_1[\alpha_{\varphi'}]\cup\varphi_2,\alpha)\\ &\geq\max_\alpha eval(\varphi'\cup\varphi'_1\cup\varphi_2,\alpha)+\max_\alpha eval(\varphi''_1,\alpha)\ \ \text{[from (1)]}\\ &\geq\max_\alpha[eval(\varphi'\cup\varphi'_1\cup\varphi_2,\alpha)+eval(\varphi''_1,\alpha)]\ \ \text{[from Prop. 4]}\\ &\geq\max_\alpha eval(\varphi'\cup\varphi'_1\cup\varphi''_1\cup\varphi_2,\alpha)\ \ \text{[from Prop. 3]}\\ &=\max_\alpha eval(\varphi,\alpha)\end{aligned}$$

□

Discussion. Similar to ID-Approx, π-Approx generates the summarization set from the frontier clauses that are not satisfied by the current solution. Consequently, the performance of the MaxSat invocation in the optimality check is similar for both π-Approx and ID-Approx. π-Approx further improves the precision of the check by applying the *retain* operator to strengthen the clauses generated. The *retain* operator does incur modest overheads when computing π-Approx. In practice, however, we find that the additional precision provided by π-Approx function improves the overall performance of the algorithm by terminating the iterative process earlier.

4.2.3 The γ-APPROX Function

While both the ID-APPROX function and the π-APPROX function only consider information on the frontier clauses, γ-APPROX further improves precision by considering information from non-frontier clauses. Similar to the π-APPROX function, γ-APPROX also generates the summarization set by applying the *retain* function on the frontier clauses violated by the current partial model. The key improvement is that γ-APPROX tries to reduce the weights of generated clauses by exploiting information from the non-frontier clauses.

Before defining the γ-APPROX function, we first introduce some definitions:

$$PV(\bigvee_{i=1}^{n} v_i \vee \bigvee_{i=1}^{m} \neg v_i') = \{v_1, ..., v_n\}, \quad PV(0) = PV(1) = \emptyset$$
$$NV(\bigvee_{i=1}^{n} v_i \vee \bigvee_{i=1}^{m} \neg v_i') = \{v_1', ..., v_m'\}, \ NV(0) = NV(1) = \emptyset$$

$$\begin{aligned}
link(v, u, \varphi) &\Leftrightarrow \exists (w, c) \in \varphi : v \in NV(c) \wedge u \in PV(c) \\
tReachable(v, \varphi) &= \{v\} \cup \{u \mid (v, u) \in R^+\}, \\
&\quad \text{where } R = \{(v, u) \mid link(v, u, \varphi)\} \\
fReachable(v, \varphi) &= \{v\} \cup \{u \mid v \in tReachable(u, \varphi)\} \\
tPenalty(v, \varphi) &= \textstyle\sum \{w \mid (w, \bigvee_{i=1}^{m} \neg v_i') \in \varphi \wedge \\
&\quad \{v_1', ..., v_m'\} \cap tReachable(v, \varphi) \neq \emptyset\} \\
fPenalty(v, \varphi) &= \textstyle\sum \{w \mid (w, \bigvee_{i=1}^{n} v_i) \in \varphi \wedge \\
&\quad \{v_1, ..., v_n\} \cap fReachable(v, \varphi) \neq \emptyset\}
\end{aligned}$$

Intuitively, $tPenalty(v, \varphi)$ overestimates the penalty incurred on the objective value of φ by setting variable v to $true$, while $fPenalty(v, \varphi)$ overestimates the penalty incurred on the objective value of φ by setting variable v to $false$.

We next introduce the γ-APPROX approximation function.

Definition 13 (γ-APPROX)**.** Given φ, $\varphi' \subseteq \varphi$, and a partial model $\alpha_{\varphi'} \in MaxSAT(\varphi')$, γ-APPROX is defined as below:

$$\begin{aligned}
&\gamma\text{-APPROX}(\varphi, \varphi', \alpha_{\varphi'}) = \{(w', c') \mid (w, c) \in \varphi \setminus \varphi' \wedge \\
&\quad Var(\{(w, c)\}) \cap Var(\varphi') \neq \emptyset \wedge \alpha_{\varphi'} \not\models c \wedge c' = \\
&\quad retain(c, Var(\varphi')) \wedge w' = reduce(w, c, \varphi, \varphi', \alpha_{\varphi'}).\}
\end{aligned}$$

We next define the $reduce$ function. Let $c'' = retain(c, Var(\varphi) \setminus Var(\varphi'))$. Then, function $reduce$ is defined as follows:

1. If $c'' = 0$, $reduce(w, c, \varphi, \varphi', \alpha_{\varphi'}) = w$.
2. If $PV(c'') \neq \emptyset$ and $NV(c'') = \emptyset$, $reduce(w, c, \varphi, \varphi', \alpha_{\varphi'}) = min(w,\ min_{v \in PV(c'')} tPenalty(v, (\varphi \setminus \varphi')[\alpha_{\varphi'}]))$.
3. If $PV(c'') = \emptyset$ and $NV(c'') \neq \emptyset$, $reduce(w, c, \varphi, \varphi', \alpha_{\varphi'}) = min(w,\ min_{v \in NV(c'')} fPenalty(v, (\varphi \setminus \varphi')[\alpha_{\varphi'}]))$.
4. If $PV(c'') \neq \emptyset$ and $NV(c'') \neq \emptyset$, $reduce(w, c, \varphi, \varphi', \alpha_{\varphi'}) = min(min(w,\ min_{v \in PV(c'')} tPenalty(v, (\varphi \setminus \varphi')[\alpha_{\varphi'}])),\ min_{v \in NV(c'')} fPenalty(v, (\varphi \setminus \varphi')[\alpha_{\varphi'}]))$.

The corresponding REFINE function is:

$$\begin{aligned}
&\gamma\text{-REFINE}(\varphi_s, \varphi, \varphi', \alpha_{\varphi'}) = \{(w, c) \mid (w, c) \in \varphi \setminus \varphi' \wedge \\
&\quad (reduce(w, c, \varphi, \varphi', \alpha_{\varphi'}), retain(c, Var(\varphi'))) \in \varphi_s\}
\end{aligned}$$

Example. Let $(\varphi, \mathcal{Q})$ where $\varphi = \{(5, x), (5, \neg x \vee y), (100, \neg y \vee z), (1, \neg z)\}$ and $\mathcal{Q} = \{x, y\}$ be a Q-MAXSAT instance. Suppose that the workset is $\varphi' = \{(5, x), (5, \neg x \vee y)\}$. By invoking a MAXSAT solver on φ', we get $\alpha_{\mathcal{Q}} = [x \mapsto 1, y \mapsto 1]$ with the objective value of φ' being 10. The only clause in $\varphi \setminus \varphi'$ that uses x or y is $(100, \neg y \vee z)$, which is not satisfied by $\alpha_{\mathcal{Q}}$. By applying the *retain* operator, γ-APPROX strengthens $(100, \neg y \vee z)$ into $(100, \neg y)$. It further transforms it into $(1, \neg y)$ via *reduce*. By comparing the optimum objective value of φ' and $\varphi' \cup \{(1, \neg y)\}$, CHECK concludes that $\varphi_{\mathcal{Q}}$ is a solution of the Q-MAXSAT instance. Indeed, $[x \mapsto 1, y \mapsto 1, z \mapsto 1]$ which is a completion of $\alpha_{\varphi'}$ is a solution of the MAXSAT formula φ. On the other hand, applying π-APPROX will fail the check due to the high weight of the clause generated via *retain*.

We explain how $reduce$ works on this example. We first generate $c'' = retain(\neg y \vee z, \{x, y, z\} \setminus \{x, y\}) = z$. Then, we compute $(\varphi \setminus \varphi')[\alpha_{\mathcal{Q}}] = \{(100, \neg y \vee z), (1, \neg z)\}[[x \mapsto 1, y \mapsto 1]] = \{(100, z), (1, \neg z)\}$. As evident, setting z to 1 only violates $(1, \neg z)$, incurring a penalty of 1 on the objective value of $(\varphi \setminus \varphi')[\alpha_{\mathcal{Q}}]$. As a result, $tPenalty(z, (\varphi \setminus \varphi')[\alpha_{\mathcal{Q}}])$ = 1, which is lower than the weight of the summarization clause $(100, \neg y)$. Hence, *reduce* returns 1 as the new weight for the summarization clause. □

To prove that γ-APPROX satisfies the specification of APPROX in Definition 2, we first prove two lemmas.

Lemma 14. *Given $(w, c) \in \varphi$, $v \in PV(c)$, and $tPenalty(v, \varphi) < w$, we construct φ' as follows:*

$$\varphi' = \varphi \setminus \{(w, c)\} \cup \{(w', c)\}, \textit{where } w' = tPenalty(v, \varphi).$$

Then we have

$$\max_{\alpha} eval(\varphi, \alpha) = \max_{\alpha'} eval(\varphi', \alpha') + w - w'.$$

Proof. We first prove a claim:

$$\exists \alpha : (eval(\varphi, \alpha) = \max_{\alpha'} eval(\varphi, \alpha')) \wedge \alpha \models c.$$

We prove this by contradiction. Suppose we can only find a model α that maximizes the objective value of φ, such that $\alpha \not\models c$. We can construct a model α^1 in the following way:

$$\alpha^1(u) = \begin{cases} \alpha(u) & \text{if } u \notin tReachable(v, \varphi), \\ 1 & \text{otherwise.} \end{cases}$$

Based on the definition of $tPenalty$, we have

$$eval(\varphi, \alpha^1) \geq eval(\varphi, \alpha) - tPenalty(v, \varphi) + w.$$

Since $w \geq tPenalty(v, \varphi)$, α^1 yields no worse objective value than α. Since $\alpha^1 \models c$ as $v \in PV(c)$ and $\alpha^1(v) = 1$, we proved the claim.

Similarly, we can show

$$\exists \alpha' : (eval(\varphi', \alpha') = \max_{\alpha''} eval(\varphi', \alpha'')) \wedge \alpha' \models c.$$

Based on the two claims, we can find $\alpha \models c$ and $\alpha' \models c$ such that

$$\max_{\alpha^1} eval(\varphi, \alpha^1) = eval(\varphi, \alpha) = w + eval(\varphi \setminus \{(w, c)\}, \alpha),$$

$$\max_{\alpha^2} eval(\varphi', \alpha^2) = eval(\varphi', \alpha') = w' + eval(\varphi' \setminus \{(w', c)\}, \alpha').$$

We next show $eval(\varphi \setminus \{(w, c)\}, \alpha) = eval(\varphi' \setminus \{(w', c)\}, \alpha')$ by contradiction. Assuming $eval(\varphi \setminus \{(w, c)\}, \alpha) > eval(\varphi' \setminus \{(w', c)\}, \alpha')$, we will have $eval(\varphi', \alpha) > eval(\varphi', \alpha')$. This is because $eval(\varphi', \alpha) = eval(\varphi' \setminus \{(w', c)\}, \alpha) + w'$ and $eval(\varphi' \setminus \{(w', c)\}, \alpha) = eval(\varphi \setminus \{(w, c)\}, \alpha)$. This contradicts with the fact that $\max_{\alpha^2} eval(\varphi', \alpha^2) = eval(\varphi', \alpha')$. Thus, we conclude $eval(\varphi \setminus \{(w, c)\}, \alpha) \leq eval(\varphi' \setminus \{(w', c)\}, \alpha')$. Similarly,we can show $eval(\varphi \setminus \{(w, c)\}, \alpha) \geq eval(\varphi' \setminus \{(w', c)\}, \alpha')$

Given $eval(\varphi \setminus \{(w, c)\}, \alpha) = eval(\varphi' \setminus \{(w', c)\}, \alpha')$, we have:

$$\begin{aligned}
&\max_{\alpha^1} eval(\varphi, \alpha^1) = eval(\varphi, \alpha) = w + eval(\varphi \setminus \{(w, c)\}, \alpha) \\
&= w + eval(\varphi' \setminus \{(w', c)\}, \alpha') = eval(\varphi', \alpha') + w - w' \\
&= \max_{\alpha^2} eval(\varphi', \alpha^2) + w - w'.
\end{aligned}$$
□

Lemma 15. *Given $(w, c) \in \varphi$, $v \in NV(c)$ and $fPenalty(v, \varphi) < w$, we construct φ' as follows:*

$$\varphi' = \varphi \setminus \{(w, c)\} \cup \{(w', c)\}, \textit{where } w' = fPenalty(v, \varphi).$$

Then we have

$$\max_{\alpha} eval(\varphi, \alpha) = \max_{\alpha'} eval(\varphi', \alpha') + w - w'.$$

Proof. Analogous to the proof to Lemma 14; we omit the details. □

We now prove that γ-APPROX satisfies the specification of APPROX in Definition 2.

Theorem 16 (γ-APPROX)**.** *γ-APPROX$(\varphi, \varphi', \alpha_{\varphi'})$ summarizes the effect of $\varphi \setminus \varphi'$ with respect to $\alpha_{\varphi'}$.*

Proof. Let $\psi = \gamma$-APPROX$(\varphi, \varphi', \alpha_{\varphi'})$. We show that γ-APPROX satisfies the specification of an APPROX function in Definition 2 by proving the inequality below:

$$\max_\alpha \, eval(\varphi' \cup \psi, \alpha) + \max_\alpha \, eval(\varphi \setminus \varphi'[\alpha_{\varphi'}], \alpha) \geq \max_\alpha \, eval(\varphi, \alpha)$$

We define φ_1 and φ_2 as follows:

$$\varphi_1 = \{(w, c) \in (\varphi \setminus \varphi') \mid Var(\{(w, c)\}) \cap Var(\varphi') \neq \emptyset\},$$
$$\varphi_2 = \{(w, c) \in (\varphi \setminus \varphi') \mid Var(\{(w, c)\}) \cap Var(\varphi') = \emptyset\}.$$

φ_1 is the set of frontier clauses and φ_2 contains rest of the clauses in $\varphi \setminus \varphi'$. We further split φ_1 into the following disjoint sets:

$$\varphi_1^1 = \{(w, c) \in \varphi_1 \mid \alpha_{\varphi'} \models c\},$$
$$\varphi_1^2 = \{(w, c) \in \varphi_1 \mid \alpha_{\varphi'} \not\models c \;\wedge\; reduce(w, c, \varphi, \varphi', \alpha_{\varphi'}) = w\},$$
$$\varphi_1^3 = \{(w, c) \in \varphi_1 \mid \alpha_{\varphi'} \not\models c \wedge w' < w \wedge \exists v \in PV(c) : w' = tPenalty(v, \varphi \setminus \varphi'[\alpha_{\varphi'}]) \wedge reduce(w, c, \varphi, \varphi', \alpha_{\varphi'}) = w'\},$$
$$\varphi_1^4 = \varphi_1 \setminus (\; \varphi_1^1 \cup \varphi_1^2 \cup \varphi_1^3).$$

Effectively, φ_1^2, φ_1^3, and φ_1^4 contain all the clauses being strengthened in γ-APPROX: φ_1^2 contains all the clauses whose weights are not reduced; φ_1^3 contains all the clauses whose weights are reduced through positive literals in them; φ_1^4 contains all the clauses whose weights are reduced through negative literals in them.

Further, we define $\hat{\varphi}_1^3$ and $\hat{\varphi}_1^4$ as below:

$$\hat{\varphi}_1^3 = \{(w', c) \mid (w, c) \in \varphi_1^3 \wedge \exists v \in PV(c) : w' = tPenalty(v, \varphi \setminus \varphi'[\alpha_{\varphi'}]) \wedge reduce(w, c, \varphi, \varphi', \alpha_{\varphi'}) = w'\},$$
$$\hat{\varphi}_1^4 = \{(w', c) \mid (w, c) \in \varphi_1^4 \wedge \exists v \in NV(c) : w' = fPenalty(v, \varphi \setminus \varphi'[\alpha_{\varphi'}]) \wedge reduce(w, c, \varphi, \varphi', \alpha_{\varphi'}) = w'\}.$$

Using Lemmas 14 and 15, we prove the following claim (1):

$$\max_\alpha eval(\varphi \setminus \varphi'[\alpha_{\varphi'}], \alpha)$$
$$= \max_\alpha eval(\varphi_1^1[\alpha_{\varphi'}] \cup \varphi_1^2[\alpha_{\varphi'}] \cup \varphi_1^3[\alpha_{\varphi'}] \cup \varphi_1^4[\alpha_{\varphi'}] \cup \varphi_2[\alpha_{\varphi'}], \alpha)$$
$$\geq \max_\alpha eval(\varphi_1^1[\alpha_{\varphi'}] \cup \varphi_1^2[\alpha_{\varphi'}] \cup \hat{\varphi}_1^3[\alpha_{\varphi'}] \cup \varphi_1^4[\hat{\alpha}_{\varphi'}] \cup \varphi_2[\alpha_{\varphi'}], \alpha)$$
$$+ \sum\{w \mid (w, c) \in \varphi_1^3 \cup \varphi_1^4\} - \sum\{w' \mid (w', c) \in \hat{\varphi}_1^3 \cup \hat{\varphi}_1^4\}$$

Similar to the soundness proof for π-APPROX, we show:

$$\pi\text{-DECOMP}(\varphi' \cup \varphi_1^2 \cup \hat{\varphi}_1^3 \cup \hat{\varphi}_1^4 \cup \varphi_2, Var(\varphi')) =$$
$$(\varphi' \cup \psi, \varphi_1^2[\alpha_{\varphi'}] \cup \hat{\varphi}_1^3[\alpha_{\varphi'}] \cup \hat{\varphi}_1^4[\alpha_{\varphi'}] \cup \varphi_2[\alpha_{\varphi'}]).$$

Following Lemma 11, we can derive the following claim (2):

$$\max_\alpha eval(\varphi' \cup \psi, \alpha) +$$
$$\max_\alpha eval(\varphi_1^2[\alpha_{\varphi'}] \cup \hat{\varphi}_1^3[\alpha_{\varphi'}] \cup \hat{\varphi}_1^4[\alpha_{\varphi'}] \cup \varphi_2[\alpha_{\varphi'}], \alpha)$$
$$\geq \max_\alpha eval(\varphi' \cup \varphi_1^2 \cup \hat{\varphi}_1^3 \cup \hat{\varphi}_1^4 \cup \varphi_2, \alpha).$$

By contradiction we can show the following claim (3) holds:

$$\forall \varphi, (w, c) \in \varphi, w' \leq w. \max_\alpha eval(\varphi, \alpha) \leq \max_\alpha eval(\varphi \setminus \{(w, c)\} \cup \{(w', c)\}, \alpha) + w - w'.$$

Combining Claim (1), (2), and (3), we have

$$\max_\alpha eval(\varphi' \cup \psi, \alpha) + \max_\alpha eval(\varphi \setminus \varphi'[\alpha_{\varphi'}], \alpha)$$
$$\geq \max_\alpha eval(\varphi' \cup \psi, \alpha) +$$
$$\max_\alpha eval(\varphi_1^1[\alpha_{\varphi'}] \cup \varphi_1^2[\alpha_{\varphi'}] \cup \hat{\varphi}_1^3[\alpha_{\varphi'}] \cup \varphi_1^4[\hat{\alpha}_{\varphi'}] \cup \varphi_2[\alpha_{\varphi'}], \alpha)$$
$$+ \sum\{w \mid (w, c) \in \varphi_1^3 \cup \varphi_1^4\} - \sum\{w' \mid (w', c) \in \hat{\varphi}_1^3 \cup \hat{\varphi}_1^4\}$$
$$\geq \max_\alpha eval(\varphi' \cup \varphi_1^2 \cup \hat{\varphi}_1^3 \cup \hat{\varphi}_1^4 \cup \varphi_2, \alpha) + \max_\alpha eval(\varphi_1^1, \alpha)$$
$$+ \sum\{w \mid (w, c) \in \varphi_1^3 \cup \varphi_1^4\} - \sum\{w' \mid (w', c) \in \hat{\varphi}_1^3 \cup \hat{\varphi}_1^4\}$$
$$\geq \max_\alpha eval(\varphi' \cup \varphi_1^1 \cup \varphi_1^2 \cup \hat{\varphi}_1^3 \cup \hat{\varphi}_1^4 \cup \varphi_2, \alpha)$$
$$+ \sum\{w \mid (w, c) \in \varphi_1^3 \cup \varphi_1^4\} - \sum\{w' \mid (w', c) \in \hat{\varphi}_1^3 \cup \hat{\varphi}_1^4\}$$
$$\geq \max_\alpha \, eval(\varphi, \alpha). \qquad \square$$

Discussion. γ-APPROX is the most precise among the three functions as it is the only one that considers the effects of non-frontier clauses. This improved precision comes at the cost of computing additional information from the non-frontier clauses. Such information can be computed in polynomial time via graph reachability algorithms. In practice, we find this overhead to be negligible compared to the performance boost for the overall approach. Therefore, in the empirical evaluation, we use γ-APPROX and its related REFINE function in our implementation.

5. Empirical Evaluation

This section evaluates our approach on Q-MAXSAT instances generated from several real-world problems in program analysis and information retrieval.

5.1 Experimental Setup

We implemented our algorithm for Q-MAXSAT in a tool PILOT. In all our experiments, we used the MiFuMaX solver [23] as the underlying MAXSAT solver, although PILOT allows using any off-the-shelf MAXSAT solver. We also study the effect of different such solvers on the overall performance of PILOT.

All our experiments were performed on a Linux machine with 8 GB RAM and a 3.0 GHz processor. We limited each invocation of the MAXSAT solver to 3 GB RAM and one hour of CPU time. Next, we describe the details of the Q-MAXSAT instances that we considered for our evaluation.

Instances from program analysis. These are instances generated from two fundamental static analyses for sequential and concurrent programs: a pointer analysis and a datarace analysis. Both analyses are expressed in a framework that combines conventional rules specified by analysis writers with feedback on false alarms provided by analysis users [35]. The framework produces MAXSAT instances whose hard clauses express soundness conditions of the analysis, while soft clauses specify false alarms identified by users. The goal is to automatically generalize user feedback to other false alarms produced by the analysis on the same input program.

The pointer analysis is a flow-insensitive, context-insensitive, and field-sensitive pointer analysis with allocation site heap abstraction [32]. Each query for this analysis is a Boolean variable that represents whether an unsafe downcast identified by the analysis prior to feedback is a true positive or not.

The datarace analysis is a combination of four analyses described in [43]: call-graph, may-alias, thread-escape, and lockset analyses. A query for this analysis is a Boolean variable that represents whether a datarace reported by the analysis prior to feedback is a true positive or not.

We generated Q-MAXSAT instances by running these two analyses on nine Java benchmark programs. Table 1 shows the characteristics of these programs. Except for the first three smaller programs in the table, all the other programs are from the DaCapo suite [10]. Table 2 shows the numbers of queries, variables, and clauses for the Q-MAXSAT instances corresponding to the above two analyses on these benchmark programs.

benchmark	brief description	# classes		# methods		bytecode (KB)		source (KLOC)	
		app	total	app	total	app	total	app	total
`ftp`	Apache FTP server	93	414	471	2,206	29	118	13	130
`hedc`	web crawler from ETH	44	353	230	2,134	16	140	6	153
`weblech`	website download/mirror tool	11	576	78	3,326	6	208	12	194
`antlr`	generates parsers and lexical analyzers	111	350	1,150	2,370	128	186	29	131
`avrora`	AVR microcontroller simulator	1,158	1,544	4,234	6,247	222	325	64	193
`chart`	plots graphs and render them as PDF	192	750	1,516	4,661	102	306	54	268
`luindex`	document indexing tool	206	619	1,390	3,732	102	235	39	190
`lusearch`	text searching tool	219	640	1,399	3,923	94	250	40	198
`xalan`	XML to HTML transforming tool	423	897	3,247	6,044	188	352	129	285

Table 1: Characteristics of the benchmark programs . Columns "total" and "app" are with and without counting JDK library code, respectively.

	datarace analysis			pointer analysis		
	# queries	# variables	# clauses	# queries	# variables	# clauses
`ftp`	338	1.2M	1.4M	55	2.3M	3M
`hedc`	203	0.8M	0.9M	36	3.8M	4.8M
`weblech`	7	0.5M	0.9M	25	5.8M	8.4M
`antlr`	0	-	-	113	8.7M	13M
`avrora`	803	0.7M	1.5M	151	11.7M	16.3M
`chart`	0	-	-	94	16M	22.3M
`luindex`	3,444	0.6M	1.1M	109	8.5M	11.9M
`lusearch`	206	0.5M	1M	248	7.8M	10.9M
`xalan`	11,410	2.6M	4.9M	754	12.4M	18.7M

Table 2: Number of queries, variables, and clauses in the MAXSAT instances generated by running the datarace analysis and the pointer analysis on each benchmark program. The datarace analysis has no queries on `antlr` and `chart` as they are sequential programs.

Instances from information retrieval. These are instances generated from problems in information retrieval. In particular, we consider problems in relational learning where the goal is to infer new relationships that are likely present in the data based on certain rules. Relational solvers such as Alchemy [29] and Tuffy [45] solve such problems by solving a system of weighted constraints generated from the data and the rules. The weight of each clause represents the confidence in each inference rule. We consider Q-MAXSAT instances generated from three standard relational learning applications, described next, whose datasets are publicly available [1, 2, 7].

The first application is an advisor recommendation system (*AR*), which recommends advisors for first year graduate students. The dataset for this application is generated from the AI genealogy project [1] and DBLP [2]. The query specifies whether a professor is a suitable advisor for a student. The Q-MAXSAT instance generated consists of 10 queries, 0.3 million variables, and 7.9 million clauses.

The second application is Entity Resolution (*ER*), which identifies duplicate entities in a database. The dataset is generated from the Cora bibliographic dataset [7]. The queries in this application specify whether two entities in the dataset are the same. The Q-MAXSAT instance generated consists of 25 queries, 3 thousand variables, and 0.1 million clauses.

The third application is Information Extraction (*IE*), which extracts information from text or semi-structured sources. The dataset is also generated from the Cora dataset. The queries in this application specify extractions of the author, title, and venue of a publication record. The Q-MAXSAT instance generated consists of 6 queries, 47 thousand variables, and 0.9 million clauses.

5.2 Evaluation Result

To evaluate the benefits of being query-guided, we measured the running time and memory consumption of PILOT. We used MiFuMaX as the baseline by running it on MAXSAT instances generated from our Q-MAXSAT instances by removing queries. To better understand the benefits of being query-guided, we also study the size of clauses posed to the underlying MAXSAT solver in the last iteration of PILOT, and the corresponding solver running time.

Further, to understand the cost of resolving each query, we pick one Q-MAXSAT instance from both domains and evaluate the performance of PILOT by resolving each query separately.

Finally, we study the sensitivity of PILOT's performance to the underlying MAXSAT solver by running PILOT using three different solvers besides MiFuMaX.

Performance of our approach vs. baseline approach. Table 3 summarizes our evaluation results on Q-MAXSAT instances generated from both domains.

Our approach successfully terminated on all instances and significantly outperformed the baseline approach in memory consumption on all instances, while the baseline only finished on twelve of the twenty instances in total. On the eight largest instances, the baseline approach either ran out of memory (exceeded 3 GB), or timed out (exceeded one hour).

Column 'peak memory' shows the peak memory consumption of our approach and the baseline approach on all instances. For the instances on which both approaches terminated, our approach consumed 55.7% less memory compared to the baseline. Moreover, for large instances containing more than two million clauses, this number further improves to 71.6%.

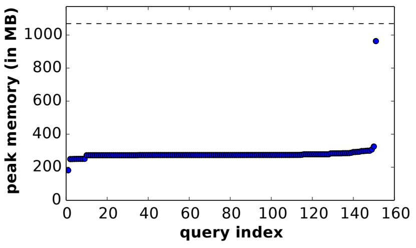
We next compare the running time of both approaches under the 'running time' column. For most instances on which both approaches terminated, our approach does not outperform the baseline in running time. One exception is *IE* where our approach terminated in 2 seconds while the baseline approach spent 2,760 seconds, yielding a 1,380X speedup. Due to the iterative nature of PILOT, on simple instances where the baseline approach runs efficiently, the overall running time of PILOT can exceed that of the baseline approach. As column 'iteration' shows, PILOT takes multiple iterations on most instances. However, for challenging instances like *IE* and other instances where the baseline approach failed to finish, PILOT yields significant improvement in running time.

To better understand the gains of being query-guided, we study the number of clauses PILOT explored under the 'problem size' column. Column 'final' shows the number of clauses PILOT posed to the underlying solver in the last iteration, while column 'max' shows the total number of clauses in the Q-MAXSAT instances. On average, PILOT only explored 35.2% clauses in each instance. Moreover, for large instances containing over two million clauses, this number improves to 19.4%. Being query-guided allows our approach to lazily explore the problem and only solve the clauses that are relevant to the queries. As a result, our approach significantly outperforms the baseline approach.

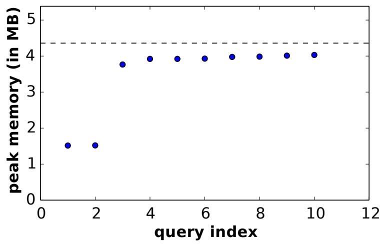
The benefit of being query-guided is also reflected by the running time of the underlying MAXSAT solver in our approach. Column 'final solver time' shows the running time of the underlying

application	benchmark	running time (in seconds)		peak memory (in MB)		problem size (in thousands)		final solver time (in seconds)	iterations
		PILOT	BASELINE	PILOT	BASELINE	final	max		
datarace	`ftp`	53	5	387	589	892	1,400	3	7
	`hedc`	45	4	260	387	586	925	2	6
	`weblech`	2	1	199	340	561	937	1	1
	`avrora`	55	5	416	576	991	1,521	2	6
	`luindex`	72	4	278	441	657	1,120	2	6
	`lusearch`	45	3	223	388	575	994	2	8
	`xalan`	145	21	1,328	1,781	3,649	4,937	15	5
pointer analysis	`ftp`	16	11	16	1,262	29	2,982	0.1	9
	`hedc`	23	21	181	1,918	400	4,821	3	7
	`weblech`	4	***timeout***	363	***timeout***	922	8,353	4	1
	`antlr`	190	***timeout***	1,405	***timeout***	3,304	12,993	14	9
	`avrora`	178	***timeout***	1,095	***timeout***	2,649	16,344	13	8
	`chart`	253	***timeout***	721	***timeout***	1,770	22,325	8	6
	`luindex`	169	***timeout***	944	***timeout***	2,175	11,882	12	8
	`lusearch`	115	***timeout***	659	***timeout***	1,545	10,917	8	9
	`xalan`	646	***timeout***	1,312	***timeout***	3,350	18,713	19	8
AR	-	4	***timeout***	4	***timeout***	2	7,968	0.3	7
ER	-	13	2	6	44	9	104	0.2	19
IE	-	2	2,760	13	335	27	944	0.05	7

Table 3: Performance of our PILOT and the baseline approach (BASELINE). In all experiments, we used a memory limit of 3 GB and a time limit of one hour for each invocation of the MAXSAT solver in both approaches. Experiments that timed out exceeded either of these limits.



(a) pointer analysis.



(b) *AR*.

Figure 4: The memory consumption of PILOT when it resolves each query separately on instances generated from (a) pointer analysis and (b) *AR*. The dotted line represents the memory consumption of PILOT when it resolves all queries together.

solver in the last iteration of PILOT. Despite the fact that the baseline approach outperforms PILOT in overall running time for some instances, the running time of the underlying solver is consistently lower than that of the baseline approach. On average, this time is only 42.1% of the time for solving the whole instance.

Effect of resolving queries separately. We studied the cost of resolving each query by evaluating the memory consumption of PILOT when applying it to each query separately. Figure 4 shows the result on one instance generated from the pointer analysis and the instance generated from *AR*. The program used in the pointer analysis is `avrora`. For comparison, the dotted line in the figure shows the memory consumed by PILOT when resolving all queries together. The other instances yield similar trends and we omit showing them for brevity.

instance	solver	running time **(in sec.)**		peak memory **(in MB)**	
		PILOT	BASELINE	PILOT	BASELINE
pointer analysis	CCLS2akms	***timeout***	***timeout***	***timeout***	***timeout***
	Eva500	2,267	***timeout***	1,379	***timeout***
	MaxHS	555	***timeout***	1,296	***timeout***
	WPM-2014.co	609	***timeout***	1,127	***timeout***
	MiFuMaX	178	***timeout***	1,095	***timeout***
AR	CCLS2akms	148	***timeout***	13	***timeout***
	Eva500	21	***timeout***	2	***timeout***
	MaxHS	4	***timeout***	9	***timeout***
	WPM-2014.co	6	***timeout***	2	***timeout***
	MiFuMaX	4	***timeout***	4	***timeout***

Table 4: Performance of our approach and the baseline approach with different underlying MAXSAT solvers.

For instances generated from the pointer analysis, when each query is resolved separately, except for one outlier, PILOT uses less than 30% of the memory it needs when all queries are resolved together. The result shows that each query is decided by different clauses in the program analysis instances. By resolving them separately, we further improve the performance of PILOT. This is in line with locality in program analysis, which is exploited by various query-driven approaches in program analysis.

For the instance generated from *AR*, we observed different results for each query. For eight queries, PILOT uses over 85% of the memory it needs when all queries are resolved together. For the other two queries, however, it uses less than 37% of that memory. After further inspection, we found that PILOT uses a similar set of clauses to produce the solution to the former eight queries. This indicates that for queries correlated with each other, batching them together in PILOT can improve the performance compared to the cumulative performance of resolving them separately.

Effect of different underlying solvers. To study the effect of the underlying MAXSAT solver, we evaluated the performance of PILOT using CCLS2akms, Eva500, MaxHS, and wpm-2014.co as the underlying solver. CCLS2akms and Eva500 were the winners in the MaxSAT'14 competition for random instances and industrial instances, respectively, while MaxHS ranked third for crafted instances (neither of the solvers performing better than MaxHS is publicly available). We used each solver itself as the baseline for comparison. In the evaluation, we used two instances from different domains, one generated from running the pointer analysis on `avrora`, and the other generated from *AR*.

Table 4 shows the running time and memory consumption of PILOT and the baseline approach under each setting. For convenience, we also include the result with MiFuMaX as the underlying solver in the table. As the table shows, except for the run with CCLS2akms on the pointer analysis instance, PILOT terminated under all the other settings, while none of the baseline approaches finished on any instance. This shows that our approach consistently gives improved performance regardless of the underlying MAXSAT solver it invokes.

6. Related Work

The MAXSAT problem is one of the optimization extensions of the SAT problem. The original form of the MAXSAT problem does not allow any hard clauses, and requires each soft clause to have a unit weight. A model of this problem is a complete assignment to the variables that maximizes the number of satisfied clauses. Two important variations of this problem are the *weighted* MAXSAT problem and the *partial* MAXSAT problem. The weighted MAXSAT problem allows each soft clause to have an arbitrary positive weight instead of limiting it to a unit weight; a model of this problem is a complete assignment that maximizes the sum of the weights of satisfied soft clauses. The partial MAXSAT problem allows hard clauses mixed with soft clauses; a model of this problem is a complete assignment that satisfies all hard clauses and maximizes the number of satisfied soft clauses. The combination of both these variations yields the *weighted partial* MAXSAT problem, which is commonly referred to simply as the MAXSAT problem, and is the problem addressed in our work.

MAXSAT solvers are broadly classified into approximate and exact. Approximate solvers are efficient but do not guarantee optimality (i.e., may not maximize the objective value) [38] or even soundness (i.e., may falsify a hard clause) [27], although it is common to provide error bounds on optimality [54]. Our solver, on the other hand, is exact as it guarantees optimality and soundness.

There are a number of different approaches for exact MAXSAT solving, including branch-and-bound based, satisfiability-based, unsatisfiability-based, and their combinations [4, 8, 20, 22, 37, 39, 41, 42, 44]. The most successful of these on real-world instances, as witnessed in annual Max-SAT evaluations [3], perform iterative solving using a SAT solver as an oracle in each iteration [4, 41]. Such solvers differ primarily in how they estimate the optimal cost (e.g., linear or binary search), and the kind of information that they use to estimate the cost (e.g. cores, the structure of cores, or satisfying assignments). Many algorithms have been proposed that perform search on either upper bound or lower bound of the optimal cost [4, 41, 42, 44], Some algorithms efficiently perform a combined search over two bounds [20, 22]. A drawback of the most sophisticated combined search algorithms is that they modify the formula using expensive Pseudo Boolean (PB) constraints that increase the size of the formula and, potentially, slow down the solver. A recent promising approach [8] avoids this problem by using succinct formula transformations that do not use PB constraints and can be applied incrementally.

Our approach is similar in spirit to the above exact approaches in aspects such as iterative solving and optimal cost estimation. But we solve a new and fundamentally different optimization problem Q-MAXSAT, which enables our approach to be demand-driven, unlike existing approaches. In particular, it enables our approach to entirely avoid exploring vast parts of a given, very large MAXSAT formula that are irrelevant to deciding the values of a (small set of) queries in some model of the original MAXSAT formula. For this purpose, our approach invokes an off-the-shelf exact MAXSAT solver on small sub-formulae of a much larger MAXSAT formula. Our approach is thus also capable of leveraging advances in solvers for the MAXSAT problem. Conversely, it would be interesting to explore how to integrate our query-guided approach into search algorithms of existing MAXSAT solvers.

The MAXSAT problem has also been addressed in the context of probabilistic logics for information retrieval [15], such as PSLs (Probabilistic Soft Logics) [28] and MLNs (Markov Logic Networks) [48]. These logics seek to reason efficiently with very large knowledge bases (KBs) of imperfect information. Examples of such KBs are the *AR*, *ER*, and *IE* applications in our empirical evaluation (Section 5). In particular, a fully grounded formula in these logics is a MaxSAT formula, and finding a model of this formula corresponds to solving the Maximum-a-Posteriori (MAP) inference problem in those logics [11, 29, 45, 46, 49]. A query-guided approach has been proposed for this problem [55] with the same motivation as ours, namely, to reason about very large MAXSAT formulae. However, this approach as well as all other (non-query-guided) approaches in the literature on these probabilistic logics sacrifice optimality as well as soundness.

In contrast, query-guided approaches have witnessed tremendous success in the domain of program reasoning. For instances, program slicing [21] is a popular technique to scale program analyses by pruning away the program statements which do not affect an assertion (query) of interest. Likewise, counter-example guided abstraction refinement (CEGAR) based model checkers [6, 12, 19] and refinement-based pointer analyses [18, 34, 50, 51] compute a cheap abstraction which is precise enough to answer a given query. However, the vast majority of these approaches target constraint *satisfaction* problems (i.e., problems with only hard constraints) as opposed to constraint *optimization* problems (i.e., problems with mixed hard and soft constraints). To our knowledge, our approach is the first to realize the benefits of query-guided reasoning for constraint optimization problems—problems that are becoming increasingly common in domains such as program reasoning [9, 30, 31, 33, 58].

7. Conclusion

We introduced a new optimization problem Q-MAXSAT that extends the well-known MAXSAT problem with queries. The Q-MAXSAT problem is motivated by the fact that many real-world MAXSAT problems pose scalability challenges to MAXSAT solvers, and the observation that many such problems are naturally equipped with queries. We proposed efficient exact algorithms for solving Q-MAXSAT instances. The algorithms lazily construct small MAXSAT sub-problems that are relevant to answering the given queries in a much larger MAXSAT problem. We implemented our Q-MAXSAT solver in a tool PILOT that uses off-the-shelf MAXSAT solvers to efficiently solve such sub-problems. We demonstrated the effectiveness of PILOT in practice on a diverse set of 19 real-world Q-MAXSAT instances ranging in size from 100 thousand to 22 million clauses. On these instances, PILOT used only 285 MB of memory on average and terminated in 107 seconds on average, whereas conventional MAXSAT solvers timed out for eight of the instances.

Acknowledgments

We thank the referees for useful feedback. This work was supported by DARPA under agreement #FA8750-15-2-0009, NSF awards #1253867 and #1526270, and a Facebook Fellowship. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright thereon.

References

[1] AI Genealogy Project. http://aigp.eecs.umich.edu.

[2] DBLP: computer science bibliography. http://http://dblp.uni-trier.de.

[3] Max-SAT Evaluation. http://www.maxsat.udl.cat/.

[4] C. Ansótegui, M. L. Bonet, and J. Levy. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196:77–105, 2013.

[5] M. Aref, B. Kimelfeld, E. Pasalic, and N. Vasiloglou. Extending datalog with analytics in LogicBlox. In *Proceedings of the 9th Alberto Mendelzon International Workshop on Foundations of Data Management*, 2015.

[6] T. Ball and S. Rajamani. The SLAM project: Debugging system software via static analysis. In *POPL*, 2002.

[7] M. Bilenko and R. J. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *SIGKDD*, 2003.

[8] N. Bjorner and N. Narodytska. Maximum satisfiability using cores and correction sets. In *IJCAI*, 2015.

[9] N. Bjorner and A.-D. Phan. νz: Maximal satisfaction with Z3. In *Proceedings of International Symposium on Symbolic Computation in Software Science (SCSS)*, 2014.

[10] S. M. Blackburn, R. Garner, C. Hoffman, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The DaCapo benchmarks: Java benchmarking development and analysis. In *OOPSLA*, 2006.

[11] A. Chaganty, A. Lal, A. Nori, and S. Rajamani. Combining relational learning with SMT solvers using CEGAR. In *CAV*, 2013.

[12] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *JACM*, 50(5), 2003.

[13] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, 1971.

[14] P. Domingos, S. Kok, D. Lowd, H. Poon, M. Richardson, and P. Singla. Markov logic. In *Probabilistic Inductive Logic Programming*. Springer, 2008.

[15] L. Getoor and B. Taskar. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

[16] A. Gracca, I. Lynce, J. Marques-Silva, and A. L. Oliveira. Efficient and accurate haplotype inference by combining parsimony and pedigree information. In *International Conference on Algebraic and Numeric Biology (ANB)*, 2010.

[17] J. Gu, P. W. Purdom, J. Franco, and B. W. Wah. Algorithms for the satisfiability (SAT) problem: A survey. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 19–152. American Mathematical Society, 1996.

[18] S. Guyer and C. Lin. Client-driven pointer analysis. In *SAS*, 2003.

[19] T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software verification with BLAST. In *SPIN Workshop on Model Checking of Software*, 2003.

[20] F. Heras, A. Morgado, and J. Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In *AAAI*, 2011.

[21] S. Horwitz, T. W. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. In *PLDI*, 1988.

[22] A. Ignatiev, A. Morgado, V. M. Manquinho, I. Lynce, and J. Marques-Silva. Progression in maximum satisfiability. In *Proceedings of the European Conference on Artificial Intelligence*, 2014.

[23] M. Janota. MiFuMax — a literate MaxSAT solver, 2013.

[24] M. Jose and R. Majumdar. Bug-assist: Assisting fault localization in ANSI-C programs. In *CAV*, 2011.

[25] M. Jose and R. Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *PLDI*, 2011.

[26] F. Juma, E. I. Hsu, and S. A. McIlraith. Preference-based planning via MaxSAT. In *Proceedings of the Canadian Conference on Artificial Intelligence*, 2012.

[27] H. Kautz, B. Selman, and Y. Jiang. A general stochastic approach to solving problems with hard and soft constraints. In *The Satisfiability Problem: Theory and Applications*, 1996.

[28] A. Kimmig, S. H. Bach, M. Broecheler, B. Huang, and L. Getoor. A short introduction to probabilistic soft logic. In *NIPS Workshop on Probabilistic Programming: Foundations and Applications*, 2012.

[29] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 2007. http://alchemy.cs.washington.edu.

[30] D. Larraz, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Proving termination of imperative programs using Max-SMT. In *FMCAD*, 2013.

[31] D. Larraz, K. Nimkar, A. Oliveras, E. Rodríguez-Carbonell, and A. Rubio. Proving non-termination using Max-SMT. In *CAV*, 2014.

[32] O. Lhoták. Spark: A flexible points-to analysis framework for Java. Master's thesis, McGill University, 2002.

[33] Y. Li, A. Albarghouthi, Z. Kincaid, A. Gurfinkel, and M. Chechik. Symbolic optimization with SMT solvers. In *POPL*, 2014.

[34] P. Liang and M. Naik. Scaling abstraction refinement via pruning. In *PLDI*, 2011.

[35] R. Mangal, X. Zhang, A. Nori, and M. Naik. A user-guided approach to program analysis. In *FSE*, 2015.

[36] R. Mangal, X. Zhang, A. Nori, and M. Naik. Volt: A lazy grounding framework for solving very large MaxSAT instances. In *SAT*, 2015.

[37] J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *DATE*, 2008.

[38] J. Marques-Silva, F. Heras, M. Janota, A. Previti, and A. Belov. On computing minimal correction subsets. In *IJCAI*, 2013.

[39] R. Martins, S. Joshi, V. M. Manquinho, and I. Lynce. Incremental cardinality constraints for MaxSAT. In *CP*, 2014.

[40] S. Miyazaki, K. Iwama, and Y. Kambayashi. Database queries as combinatorial optimization problems. In *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications (CODAS)*, 1996.

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

[42] A. Morgado, C. Dodaro, and J. Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In *CP*, 2014.

[43] M. Naik, A. Aiken, and J. Whaley. Effective static race detection for Java. In *PLDI*, 2006.

[44] N. Narodytska and F. Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In *AAAI*, 2014.

[45] F. Niu, C. Ré, A. Doan, and J. W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. In *VLDB*, 2011.

[46] J. Noessner, M. Niepert, and H. Stuckenschmidt. RockIt: Exploiting parallelism and symmetry for MAP inference in statistical relational models. In *AAAI*, 2013.

[47] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

[48] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62(1-2), 2006.

[49] S. Riedel. Improving the accuracy and efficiency of MAP inference for Markov Logic. In *UAI*, 2008.

[50] M. Sridharan and R. Bodík. Refinement-based context-sensitive points-to analysis for Java. In *PLDI*, 2006.

[51] M. Sridharan, D. Gopan, L. Shan, and R. Bodík. Demand-driven points-to analysis for Java. In *OOPSLA*, 2005.

[52] D. M. Strickland, E. R. Barnes, and J. S. Sokol. Optimal protein structure alignment using maximum cliques. *Operations Research*, 53(3):389–402, 2005.

[53] M. Vasquez and J. Hao. A ”logic-constrained” knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Comp. Opt. and Appl.*, 20(2):137–157, 2001.

[54] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[55] W. Y. Wang, K. Mazaitis, N. Lao, T. M. Mitchell, and W. W. Cohen. Efficient inference and learning in a large knowledge base: Reasoning with extracted information using a locally groundable first-order probabilistic logic. *Machine Learning Journal*, 2015.

[56] H. Xu, R. A. Rutenbar, and K. A. Sakallah. sub-SAT: a formulation for relaxed boolean satisfiability with applications in routing. In *Proceedings of the International Symposium on Physical Design (ISPD)*, 2002.

[57] Q. Yang, K. Wu, and Y. Jiang. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence*, 171(2-3):107–143, Feb. 2007.

[58] X. Zhang, R. Mangal, R. Grigore, M. Naik, and H. Yang. On abstraction refinement for program analyses in Datalog. In *PLDI*, 2014.