Constraints (2016) 21:277–302
DOI 10.1007/s10601-015-9195-9

Quantified maximum satisfiability

Alexey Ignatiev[1,2] · Mikoláš Janota[1] · Joao Marques-Silva[1,3]

Published online: 24 May 2015
© Springer Science+Business Media New York 2015

Abstract In recent years, there have been significant improvements in algorithms both for Quantified Boolean Formulas (QBF) and for Maximum Satisfiability (MaxSAT). This paper studies an optimization extension of QBF and considers the problem in a quantified MaxSAT setting. More precisely, the general QMaxSAT problem is defined for QBFs with a set of soft clausal constraints and consists in finding the largest subset of the soft constraints such that the remaining QBF is true. Two approaches are investigated. One is based on relaxing the soft clauses and performing an iterative search on the cost function. The other approach, which is the main contribution of the paper, is inspired by recent work on MaxSAT, and exploits the iterative identification of unsatisfiable cores. The paper investigates the application of these approaches to the two concrete problems of computing smallest minimal unsatisfiable subformulas (SMUS) and smallest minimal equivalent subformulas (SMES), decision versions of which are well-known problems in the second level of the polynomial hierarchy. Experimental results, obtained on representative problem instances, indicate that the core-guided approach for the SMUS and SMES problems outperforms the use of iterative search over the values of the cost function. More significantly,

✉ Alexey Ignatiev
aign@sat.inesc-id.pt

Mikoláš Janota
mikolas@sat.inesc-id.pt

Joao Marques-Silva
jpms@ucd.ie

[1] INESC-ID, IST, University of Lisbon, Lisbon, Portugal

[2] ISDCT SB RAS, Irkutsk, Russia

[3] University College Dublin, Dublin, Ireland

the core-guided approach to SMUS also outperforms the state-of-the-art SMUS extractor Digger.

Keywords Quantified Boolean formula · Optimization · SAT · MaxSAT · Smallest MUS/MES

1 Introduction

When reasoning about quantified Boolean formulas (QBF), different optimization problems can be envisioned. MAX-QSAT [21] is a well-known example. Considering a QBF as a game between the existential and universal players, if the existential player can guarantee that k clauses are satisfied independently of the universal player, then k clauses are said to be *simultaneously satisfiable*. The MAX-QSAT problem is to find the maximum number of simultaneously satisfiable clauses. Original interest in MAX-QSAT was motivated by work on non-approximability results for problems in the polynomial hierarchy. A different optimization problem is to select a subset of clauses of a QBF such that the resulting QBF is true. A related (restricted) optimization problem assumes the first quantifier to be existential, and asks for an assignment to those existential variables such that the QBF is true and a cost function is optimized. Work on quantified CSP involves computing strategies that optimize some cost function or associating costs with strategies [15, 19]. Besides the theoretical interest, there are a number of practical settings where quantified optimization problems find application. This is for example the case when the goal is to optimize a cost function subject to a quantified set of constraints (e.g. the iterative use of QBF for optimizing target features in Boolean function decomposition in [18]). Many other concrete examples are given by the optimization versions of decision problems in the polynomial hierarchy [64].

This paper addresses the problem of optimizing a cost function subject to a quantified set of constraints. The cost function will be represented as a set of *soft* clauses, and so this problem is referred to as Quantified MaxSAT (QMaxSAT). Inspired by algorithms for the non-quantified MaxSAT problem [4, 5, 16, 24, 31, 55], this paper develops two novel approaches for QMaxSAT. The first one consists of relaxing all clauses and performing a linear (or binary) search over the values of the cost function. The linear search can either refine upper or lower bounds on the number of falsified soft clauses [16, 24]. In contrast, binary search refines both lower and upper bounds [24, 31]. The second approach represents the main contribution of this paper, and is inspired by recent work on core-guided MaxSAT, i.e. solving MaxSAT by iteratively computing unsatisfiable subformulas [24]. Thus, this new approach requires QBF solvers to be able to produce unsatisfiable cores. As a result, the second contribution of this paper is to show how recent 2QBF solvers based on abstraction refinement [37, 38] can be modified to produce unsatisfiable cores.

The new algorithms for QMaxSAT are evaluated on the problems of computing the *smallest minimally unsatisfiable subformula* (SMUS) [29, 45, 49, 53] and the *smallest minimally equivalent subformula* (SMES) [11, 12, 43, 44]. The SMUS decision problem is well-known to be in the second level of the polynomial hierarchy (e.g. [29]) and studied in the context of artificial intelligence and formal verification [1, 2, 35]. Computing SMUSes is also relevant for assessing the quality of computed MUSes in practice. The SMES problem, while being a generalization of SMUS, is tightly related to detection of CNF subformula redundancy. Therefore, SMES is of great importance in the context of CNF formula minimization [11, 12, 43, 44]. Besides the SMUS/SMES problems studied in the paper, a number

of other important practical applications (e.g. see [60]) can be formulated and solved in the context of the proposed QMaxSAT framework.

The third contribution of the paper is a novel QMaxSAT formulation for the SMUS and SMES problems, and QMaxSAT-based algorithms. Experimental results, obtained on representative problem instances, show that the core-guided QMaxSAT algorithm outperforms Digger, a state-of-the-art algorithm for the SMUS problem [45]. More importantly, these results validate the use of core-guided approaches for QMaxSAT. First experimental results for the SMES problem also demonstrate the power of core-guided search in the QMaxSAT setting.

This paper extends the conference version [33] as follows. The general QMaxSAT problem is introduced, and related with the special case studied in this paper. The paper also gives a description of the new iterative QMaxSAT algorithms, extends the method of extracting unsatisfiable cores with CEGAR-based QBF for formulas with an arbitrary number of quantification levels, as well as formulates the smallest MES problem as the quantified MaxSAT problem and solves it using the proposed algorithms.

The paper is organized as follows. The next section overviews basic definitions on SAT, MaxSAT, and QBF. Section 3 introduces the QMaxSAT problem and Section 4 proposes several algorithms for QMaxSAT with an arbitrary number of quantification levels. This is complemented by a description in Section 4.2 of how unsatisfiable cores can be generated by CEGAR-based QBF solvers and like so used in QMaxSAT algorithms. Section 5 shows the practicality of the framework: it models two known problems — SMUS and SMES — as QMaxSAT and describes improvements to the QMaxSAT for the concrete problems of computing an SMUS and SMES. Section 6 presents the experimental results on computing SMUSes/SMESes. Section 7 concludes the paper.

2 Preliminaries

This section provides the notation and basic definitions related to SAT, MaxSAT, and QBF.

2.1 Boolean satisfiability

Let us consider a set of Boolean variables $X = \{x_1, \ldots, x_n\}$, $n \in \mathbb{N}$. A *literal* for variable x_i, $i \in \{1, \ldots, n\}$, is a formula denoted by l_i, which can be either a *positive* literal x_i, or its negation $\neg x_i$. A set of literals connected by a disjunction is called a *clause*. A conjunction of clauses $\varphi = c_1 \wedge c_2 \wedge \ldots \wedge c_m$, $m \in \mathbb{N}$, is called a formula in *conjunctive normal form* (CNF formula). Whenever convenient, a CNF formula is treated as a set of sets of literals $\varphi = \{c_1, c_2, \ldots, c_m\}$.

An assignment is a total mapping $\mathcal{A}_X : X \to \{0, 1\}$ defined on set X of variables. When it is clear from the context what set the mapping is defined on, the corresponding subscript X is omitted (in this case we write $\mathcal{A}$). The notion of assignment $\mathcal{A}$ can be extended to literals by setting $\mathcal{A}(\neg x_i) = 1 - \mathcal{A}(x_i)$ for $x_i \in X$. Hereinafter, expression $\varphi|_{\mathcal{A}}$ denotes a formula obtained from a Boolean formula φ by replacing each variable x_i of X with its value $\mathcal{A}(x_i)$. The same *restriction* notation $c|_{\mathcal{A}}$ is used with regard to a clause c of a CNF formula.

If $\mathcal{A}(l_i) = 1$ then literal l_i is said to be *satisfied* by assignment $\mathcal{A}$; if $\mathcal{A}(l_i) = 0$ then l_i is *falsified* by $\mathcal{A}$. Assignment $\mathcal{A}$ satisfies a clause c if it satisfies at least one of its literals; otherwise (i.e. if $c|_{\mathcal{A}} = 0$), the clause is said to be falsified by $\mathcal{A}$. If for a given CNF formula

φ there is an assignment $\mathcal{A}$ such that $\varphi|_{\mathcal{A}} = 1$, then formula φ is called *satisfiable* and $\mathcal{A}$ is a *satisfying assignment* (or a *model*). In the remainder of the paper the set of all models of a CNF formula φ is denoted by $\mathcal{M}(\varphi)$.

2.2 Maximum satisfiability

The *Maximum Satisfiability* (MaxSAT) is an optimization generalization of SAT formulated as follows: for a given CNF formula $\varphi = \{c_1, c_2, \ldots, c_m\}$, $m \in \mathbb{N}$, find a subformula φ^*, $\varphi^* \subseteq \varphi$, of the maximum possible size such that φ^* is satisfiable. The MaxSAT problem can also be formulated as finding such an assignment $\mathcal{A}$ that satisfies the maximum number of clauses of φ.

The *partial* MaxSAT problem generalizes MaxSAT and deals with CNF formulas of the form $\varphi = \varphi_S \cup \varphi_H$, where all the clauses of φ_S are declared to be *relaxable* or *soft* while the rest (clauses of φ_H) are declared to be *hard*. The problem is to find an assignment $\mathcal{A}$ that satisfies all the hard clauses and maximizes the number of the soft clauses that are satisfied. *Weighted* (*weighted partial*) MaxSAT is defined for weighted clauses (soft clauses) and consists in satisfying a subset of clauses (soft clauses) with the maximum total weight.

2.3 Quantified Boolean formula

Quantified Boolean formulas (QBFs) are an extension of propositional logic with *existential* and *universal* quantifiers ($\forall$, $\exists$) [39].

In this paper QBFs are assumed to be in *prenex closed* form $Q_1 x_1 \ldots Q_n x_n.\ \varphi$, where $Q_i \in \{\forall, \exists\}$, x_i are distinct Boolean variables, and φ is a Boolean formula over the variables x_i and the constants 0 (false), 1 (true). The sequence of quantifiers in a QBF is called the *prefix* and the Boolean formula the *matrix*. The semantics of QBF is defined recursively. A QBF $\exists x_1 Q_2 x_2 \ldots Q_n x_n.\ \varphi$ is true iff $Q_2 x_2 \ldots Q_n x_n.\ \varphi|_{x_1=1}$ or $Q_2 x_2 \ldots Q_n x_n.\ \varphi|_{x_1=0}$ is true. A QBF $\forall x_1 Q_2 x_2 \ldots Q_n x_n.\ \varphi$ is true iff both $Q_2 x_2 \ldots Q_n x_n.\ \varphi|_{x_1=1}$ and $Q_2 x_2 \ldots Q_n x_n.\ \varphi|_{x_1=0}$ are true. To decide whether a given QBF is true or not, is PSPACE-complete [39].

Within a prefix, two adjacent quantifiers of different type, namely $\forall x_i \exists x_{i+1}$ and $\exists x_i \forall x_{i+1}$, are called a *quantifier alternation*. A QBF with k alternations has $k+1$ *quantification levels*. Whenever possible, for variables $x_1, \ldots, x_n$, $x_i \in X_j$, under the same quantifier Q_j we write $Q_j X_j$ instead of $Q_j x_1 \ldots Q_j x_n$. Hence, a formula with k quantification levels takes the form $Q_1 X_1 \ldots Q_k X_k.\ \varphi$. A prefix $Q_1 X_1 \ldots Q_k X_k$ of a QBF with k quantification levels is denoted by $\overrightarrow{Q}$.

In Section 5, devoted to the SMUS and SMES problems, we focus on QBFs with two levels of quantification, i.e. formulas of the form $\forall X \exists Y.\ \varphi$ or $\exists X \forall Y.\ \varphi$, commonly denoted by 2QBF. Deciding whether a formula in 2QBF is true is complete for the second level of the polynomial hierarchy [39].

Section 3 uses the notion of *solution* of QBFs of the form $\psi = \exists X_0\, \overrightarrow{Q}.\ \varphi$. An assignment $\mathcal{A}_{X_0}$ is a solution of ψ iff $\overrightarrow{Q}.\ \varphi|_{\mathcal{A}_{X_0}}$ is true.[1] Analogously to the set of all models of a CNF formula, the set of all solutions of a QBF ψ, where the first quantifier is $\exists$, is denoted by $\mathcal{M}(\psi)$.

[1] Note that a solution of a quantified formula defined in this way represents a "portion" of the formula's *model*, which is defined, for example, in [41].

3 Quantified MaxSAT

Although QBF has a significantly more expressive power than formulas without quantifiers and, thus, can find a number of relevant applications, many practically important problems are formulated as optimization problems. So sometimes it is not enough to encode a problem into a QBF — sometimes it is needed to optimize with respect to a QBF. However, to the best of our knowledge, work in this area is lacking. In this section, we introduce an optimization formulation of the QBF problem, which is a natural generalization of MaxSAT. Instead of CNF formulas, we consider quantified formulas with a matrix specified in a general form. Definition 1 gives a precise definition of the problem studied in this paper.

Definition 1 Given a QBF

$$\overrightarrow{Q}\,.\,\varphi_H \wedge \varphi_S \tag{1}$$

whose matrix is a conjunction of a non-clausal formula φ_H and a CNF formula φ_S, the *quantified Maximum Satisfiability* (*quantified MaxSAT*, *QMaxSAT*) problem consists in finding a set of clauses φ_S^*, $\varphi_S^* \subseteq \varphi_S$, of maximum possible size such that QBF $\overrightarrow{Q}\,.\,\varphi_H \wedge \varphi_S^*$ is true. Here formulas φ_H and φ_S are called the hard and soft parts of QBF (1).

Note that QBF (1) is not required to be false; if it is true, then $\varphi_S^* = \varphi_S$. Also observe that if there is no hard part φ_H in (1) and $\overrightarrow{Q} = \exists X$, the QMaxSAT problem becomes the classical non-quantified MaxSAT. Additionally, although the problem defined above is referred to as quantified MaxSAT, it is a quantified version of the Partial MaxSAT problem since QBF (1) contains both hard and soft parts.[2] Omitting the hard part φ_H would result in a more restricted problem definition, which would be a quantified version of the classical non-partial MaxSAT.

Observe that requiring the hard part to be in an non-CNF form is essential for efficiency reasons.[3] Although any Boolean formula can be *transformed* to CNF using auxiliary Tseitin variables [59, 68], Tseitin transformation is known to be detrimental to QBF solvers' performance (e.g. see [6, 27, 42]). A number of approaches [27, 28, 72] were developed to address this issue. Currently, some of the state-of-the-art QBF solvers (e.g. [26, 42]) require non-CNF input formulas. Moreover, many QBF (and even SAT) instances come from Boolean circuits and other domains where original problem instances are not in CNF [66]. Thus, knowing the structure of the original problem is usually crucial for solving algorithms. Therefore, QBF algorithms described in the paper are supposed to Tseitin-encode the hard part of the considered QBFs only when the calls to a SAT oracle are performed.

Example 1 Consider a 2QBF

$$\exists x_1, x_2\, \forall y_1, y_2.\, \varphi_H \wedge \varphi_S$$

where the hard part $\varphi_H = (\neg x_1 \wedge \neg x_2) \rightarrow (y_1 \vee y_2)$ and the soft part $\varphi_S = \{(\neg x_1), (\neg x_2), (\neg x_1 \vee \neg x_2), (x_1 \vee \neg x_2)\}$. Obviously, the considered QBF is false. Among

[2] One can also consider clauses of φ_S to be weighted, which would result in the weighted quantified MaxSAT problem formulation.

[3] And both applications considered in Section 5 indeed have a non-CNF hard part.

all the subsets of φ_S, the largest one is the solution of the QMaxSAT problem for the QBF, and it is $\varphi_S^* = \{(\neg x_2), (\neg x_1 \vee \neg x_2), (x_1 \vee \neg x_2)\}$.

In order to solve the general formulation of the QMaxSAT problem defined above, it is convenient to use the following restricted formalism, which reformulates the problem by making use of *selection variables* and enables us to use the notion of a QBF solution.

Proposition 1 *The QMaxSAT problem for QBF* (1) *can be reduced to the QMaxSAT problem for QBF of the following form*

$$\exists S \overrightarrow{Q} . \phi_H \wedge \phi_S \tag{2}$$

where ϕ_H is a Boolean non-CNF formula while ϕ_S is a set of unit clauses over selection variables S. Given a QMaxSAT solution ϕ_S^* for (2)*, one can reconstruct the QMaxSAT solution φ_S^* for* (1)*. Both ϕ_S^* and φ_S^* are defined by the corresponding assignment $\mathcal{A}_S^*$ to selection variables S.*

Proof Let us explicitly construct QBF (2) for the original QBF (1). Consider formula $\overrightarrow{Q} . \varphi_H \wedge \varphi_S$, where φ_H is the hard part in a potentially non-CNF form and φ_S is a set of clauses. Each clause $c \in \varphi_S$ can be extended to $c \vee \neg s$, where variable s is called a selection variable for clause c. Selection variables act as flags determining whether the corresponding clauses of φ_S are selected or not, i.e. by assigning variable s to true we force the original clause c to be satisfied; otherwise, it is deselected since literal $\neg s$ is true. Let us denote the set of all selection variables by S. After extending original clauses of φ_S this way, we make all of them hard. Let us introduce a new hard part of the QBF being constructed $\phi_H = \varphi_H \wedge \{c \vee \neg s \mid c \in \varphi_S\}$. A new soft part of the QBF matrix is $\phi_S = \{s \mid s \in S\}$. CNF ϕ_S contain only unit soft clauses defined on the selection variables S. Thus, QBF $\exists S \overrightarrow{Q} . \phi_H \wedge \phi_S$ is constructed.

Now the quantified MaxSAT problem is to find a subset of soft clauses ϕ_S^*, $\phi_S^* \subseteq \phi_S$, of the largest possible size such that QBF $\overrightarrow{Q} . \phi_H \wedge \phi_S^*$ is true. This essentially means that one needs to find an assignment $\mathcal{A}_S^*$ that defines this largest subset ϕ_S^*. By construction of QBF $\exists S \overrightarrow{Q} . \phi_H \wedge \phi_S$, assignment $\mathcal{A}_S^*$ also defines the QMaxSAT solution for the original QBF (1). □

Example 2 Consider the QBF from Example 1. The modification of the formula results in introducing 4 selection variables, one per soft clause. The new QBF is $\exists s_1, s_2, s_3, s_4, x_1, x_2 \, \forall y_1, y_2 . \phi_H \wedge \phi_S$, where $\phi_H = \varphi_H \wedge ((\neg x_1 \vee \neg s_1) \wedge (\neg x_2 \vee \neg s_2)(\neg x_1 \vee \neg x_2 \vee \neg s_3) \wedge (x_1 \vee \neg x_2 \vee \neg s_4))$ and $\phi_S = \{(s_1), (s_2), (s_3), (s_4)\}$. The QMaxSAT solution is assignment $\mathcal{A}_S^* = \{\neg s_1, s_2, s_3, s_4\}$.

It should be noted that the following optimization of QBF (2) can be done if some of the clauses of the original soft CNF formula φ_S are *unit*, i.e. they have just one literal. Indeed, there is no need to introduce a selection variable for a *unit* soft clause. Unit soft clauses are not made hard and are preserved in ϕ_S without changes.

As one can notice, reformulating QMaxSAT in the described way can increase the number of quantification levels of the considered QBF. This is the case if prefix $\overrightarrow{Q}$ of (1) starts with a universal quantifier $\forall$, which should be preceded by a new existential quantifier $\exists S$. However, this is not a problem for QBFs whose prefix already starts with $\exists$. In any case, despite this issue, the considered formalism enables us to use a unified approach to the

QMaxSAT problem for QBFs whose prefix starts with both ∀ and ∃. Therefore, from now on we are going to consider and solve the QMaxSAT problem for only QBFs of the form (2).

Due to the close relationship of the QMaxSAT problem to its classical version, an interesting line of work is to apply to this problem the ideas and algorithms developed for non-quantified MaxSAT. The next section gives a detailed explanation of how MaxSAT algorithms can be adapted to the QMaxSAT problem.

3.1 Related work

Optimization problems subject to quantified constraints have been studied elsewhere [15, 19, 21], but address more general formulations than QMaxSAT. The Max-QSAT problem [21] can be viewed as computing a strategy that maximizes the number of simultaneously satisfiable clauses. Other optimization problems have been studied in the recent past [15, 19]. The focus of [19] is approximation algorithms for computing a winning strategy that minimizes some cost function, whereas [15] studies preferences over strategies. A framework similar to QMaxSAT was studied in [60] in the context of the QBF prefix optimization problem, where only an iterative search algorithm was considered. To our best knowledge, and besides our work, [15] and [60] are the only other references that propose exact algorithms for solving optimization problems over quantified constraints.

4 Solving QMaxSAT

Analogously to what was done for a number of other optimization problems, solving quantified MaxSAT requires developing specific iterative procedures. This section proposes several algorithms for solving QMaxSAT. Being based on the algorithms developed for the classical MaxSAT problem, all the algorithms described in this section hinge upon the idea of iterative use of QBF oracles. The main focus of the section is the algorithm based on unsatisfiable core extraction. Thus, and in addition to the QMaxSAT algorithms, the section defines a notion of an unsatisfiable core for QBF formulas and also gives a detailed explanation of how they can be extracted in modern QBF solvers.

4.1 QMaxSAT algorithms

4.1.1 Iterative QMaxSAT algorithms

This section describes one of the simplest approaches to the QMaxSAT problem also used in [60]. Recall that we are going to solve the quantified MaxSAT problem for QBFs of the form (2), which consists in finding the best solution $\mathcal{A}^*_S$ of the hard part of QBF (2) (if any) with respect to the number of satisfied clauses in ϕ_S, i.e. $|\phi^*_S|$ where ϕ^*_S is the largest subset of ϕ_S whose clauses are satisfied by $\mathcal{A}^*_S$. Analogously to classical non-quantified MaxSAT, we will optimize on the *cost* of solution ϕ^*_S, i.e. the number of falsified clauses in ϕ_S, which is $|\phi_S| - |\phi^*_S|$. Note that it is not hard to find both upper and lower bounds for the solution cost, at least the trivial ones.[4] Let us denote upper and lower bounds on the cost as μ and λ, respectively.

[4] The trivial lower and upper bounds are 0 and $|\phi_S|$, respectively.

The idea of the approach is to iteratively choose a value k from $\{\lambda, \lambda+1, \ldots, \mu-1, \mu\}$ and to decide if there is a solution of the hard part of (2) that falsifies at most k clauses of ϕ_S. Observe that the basic idea is essentially to consider and constrain the value $\sum_{s\in S} \neg s \leq k$ and check if there is a solution of the hard part of (2) that respects this constraint.[5] This kind of constraints on $\sum_{s\in S} \neg s$ can be easily encoded into a Boolean formula, e.g. into a formula in conjunctive normal form [62]. Assuming such constraints to be in CNF, the considered QMaxSAT problem can be solved by successive deciding whether the following QBF is true or false:

$$\exists S \overrightarrow{Q}. \phi_H \wedge (\sum\nolimits_{s\in S} \neg s \leq k) \tag{3}$$

There are different possible strategies on how to choose value k. For example, one can start with $k = \lambda$ and increment k until formula (3) becomes true, or decrease it starting from the upper bound ($k = \mu$) while (3) is true. This is analogous to the linear search for non-quantified MaxSAT [16], which refines lower and upper bounds on the value of the cost function.[6]

As a brief example, the pseudo-code of the linear search UNSAT-SAT (LSUS, which was originally developed for MaxSAT) adapted to quantified MaxSAT is shown in Algorithm 1. The algorithm is referred to as *QLSUS* (*quantified LSUS*). Given formula (2), it finds a solution $\mathcal{A}^*_S$ of its hard part that is *the best* with respect to the number of satisfied clauses in ϕ_S, i.e. with the smallest cost. The pre-condition of the algorithm requiring the hard part of the QBF to be true ensures that the algorithm terminates. The algorithm initializes k with value 0. At each iteration of the loop, QBF (3), where constraint $\sum_{s\in S} \neg s \leq k$ is encoded into a CNF formula, is decided by a QBF oracle. If the formula is false, algorithm QLSUS increases k, updates the QBF accordingly and continues doing the loop. Otherwise, the algorithm stops and reports the solution found. Observe that it is not hard to modify the algorithm for implementing linear search SAT-UNSAT or binary search (the corresponding acronyms would be *QLSSU* and *QBINS*, respectively). Although all these iterative algorithms are not the main contribution of the paper, we implemented and compared them to our main algorithm for the concrete case of the SMUS problem (see Section 6).

Algorithm 1 An example of an iterative QMaxSAT algorithm: QLSUS

input : A QBF $\psi = \exists S \overrightarrow{Q}. \phi_H \wedge \phi_S$ s.t. $\exists S \overrightarrow{Q}. \phi_H$ is true, and ϕ_S is defined on S
output : QMaxSAT solution $\mathcal{A}^*_S \in \mathcal{M}(\psi)$

```
1 k ← 0                                                    // lower bound initialization
2 while true:
3     (st, A_S) ← QBF(∃S Q⃗. φ_H ∧ CNF(Σ_{s∈S} ¬s ≤ k))      // calling a QBF oracle
4     if st = true:
5         return A_S
6     k ← k + 1                                            // updating lower bound
```

4.1.2 Core-guided QMaxSAT algorithms

The main goal of this paper is to construct an algorithm which is based on the use of *unsatisfiable cores* (or simply *cores*) similar to the Fu&Malik's algorithm for MaxSAT [24].

[5] Note that the soft part ϕ_S of QBF (2) forces the fact that $\sum_{s\in S} \neg s = 0$.

[6] Instead of the linear search algorithms, one can use binary search [24, 31]. Binary search algorithms are not covered by this paper.

Although there are a number of algorithms that are significantly more efficient in practice[7] (e.g. MSU3 and BCD2 [55], Progression [34], OLL [54], and MaxRes [56]), we mainly focus on the Fu&Malik's algorithm since it was the first proposed algorithm based on the idea of unsatisfiable core extraction and it is relatively easy to implement. However, note that all the MaxSAT algorithms can be adapted to QMaxSAT using the ideas proposed in this section. Similarly to the linear search that refines lower bounds on the value of the cost function, Fu&Malik's algorithm (we refer to its original version as MSU1 [50]; some authors refer to this algorithm as WPM1 [3]) tests a series of unsatisfiable instances until a satisfiable instance is found. However, instead of dealing with the constraint $\sum_{s \in S} \neg s \leq k$ and increasing k with each call to a SAT solver, it identifies a small unsatisfiable portion of the soft part ϕ_S, which is called an unsatisfiable core. Sequential core computation in MSU1 increases the current cost value with each iteration, i.e. with every new core computed. Thus, each unsatisfiable core increments a possible minimum cost of an assignment that satisfies the constraints. Let us define a core of formula (2). This will enable us to extend the MSU1 algorithm to the case of QMaxSAT.

Definition 2 Given QBF (2), a Boolean formula $\phi_H \wedge \phi'_S$, where $\phi'_S \subseteq \phi_S$, is called an unsatisfiable core of formula (2), if and only if QBF $\exists S \, \overrightarrow{Q} . \, \phi_H \wedge \phi'_S$ is false.

Even though Definition 2 is made for the reformulated QBF (2), observe that one can analogously define unsatisfiable cores for the original QBFs of the form (1). Also notice that according to Definition 2, the hard part ϕ_H of QBF (2) is included into any unsatisfiable core of the QBF. However, similarly to the core-guided algorithms for the non-quantified MaxSAT, the algorithm described below needs only the soft part ϕ'_S of the core while the hard part is not used and, thus, not needed. Therefore, hereinafter we assume that all the following algorithms are able to implicitly extract and deal with the soft part ϕ'_S of a QBF core.

Algorithm 2 QMSU1 Algorithm

input : A QBF $\psi = \exists S \, \overrightarrow{Q} . \, \phi_H \wedge \phi_S$ s.t. ϕ_S is a set of clauses over S
output : QMaxSAT solution $\mathcal{A}^*_S \in \mathcal{M}(\psi)$ if any; false otherwise

1 $R_{\text{all}} \leftarrow \emptyset$ // set of all relaxation variables
2 **while** true:
3 $\quad \psi_W = \exists S \, \exists R_{\text{all}} \, \overrightarrow{Q} . \, \phi_H \wedge \phi_S$
4 $\quad (\text{st}, \phi'_S, \mathcal{A}_S) \leftarrow \texttt{QBF}(\psi_W)$ // calling a QBF oracle
5 $\quad$ **if** st = true:
6 $\quad\quad$ **return** $\mathcal{A}_S$ // found solution $\mathcal{A}^*_S = \mathcal{A}_S$
7 $\quad$ **if** $\phi'_S = \emptyset$:
8 $\quad\quad$ **return** false // $\exists S \, \overrightarrow{Q} . \, \phi_H$ is false
9 $\quad R \leftarrow \emptyset$ // set of relaxation variables
10 $\quad$ **foreach** $c \in \phi'_S$:
11 $\quad\quad$ **let** r be a new relaxation variable
12 $\quad\quad R \leftarrow R \cup \{r\}$
13 $\quad\quad \phi_S \leftarrow \phi_S \setminus \{c\} \cup \{c \vee r\}$
14 $\quad \phi_H \leftarrow \phi_H \wedge \texttt{CNF}(\sum_{r \in R} r \leq 1)$
15 $\quad R_{\text{all}} \leftarrow R_{\text{all}} \cup R$

[7] http://www.maxsat.udl.cat/14/

Algorithm 2 shows the pseudo-code of the MSU1 algorithm adapted to QMaxSAT (we refer to this algorithm as *QMSU1*, *quantified MSU1*). Given a formula ψ in the form (2), the QMSU1 algorithm computes such a solution $\mathcal{A}_S^*$ of the hard part of ψ (if any) that maximizes the number of satisfied clauses of ϕ_S. The set of all relaxation variables used by the algorithm is denoted by R_{all} and initialized by $\emptyset$ (line 1). At each iteration of the loop the algorithm constructs a working copy ψ_W of formula ψ (3) and asks a QBF oracle to decide whether it is true or false (4). Notice that this working copy of the formula is modified at each iteration of the algorithm since new relaxation variables are introduced for each new unsatisfiable core. As an answer the oracle returns a 3-tuple $(\mathsf{st}, \phi'_S, \mathcal{A}_E)$ (line 4). If $\mathsf{st} = \mathsf{false}$, then a new unsatisfiable core returned by the oracle is processed. If the soft part ϕ'_S of the core is empty, it means that the hard part ϕ_H of the QBF is itself inconsistent, i.e. $\exists S \overrightarrow{Q} . \phi_H$ is false, and the algorithm returns false. Otherwise (if the core is not empty), the algorithm considers a set of relaxation variables R (initially set to $\emptyset$) and relaxes each soft clause $c \in \phi'_S$ of the core by a new relaxation variable of $r \in R$. The new (extended) soft clause $c \vee r$ replaces the original clause c in ϕ_S. At the end of the iteration, QMSU1 adds a CNF encoding of a new cardinality constraint $\sum_{r \in R} r \leq 1$ to the hard part ϕ_H of ψ. Note that since every clause of ϕ_S contains only selection or (previously added) relaxation variables, all the relaxation variables $r \in R_{\text{all}}$ can be quantified by the same $\exists$-quantifier as variables $s \in S$ (see line 3). The algorithm iterates until formula ψ_W is true, in which case it reports the solution $\mathcal{A}_S \in \mathcal{M}(\psi_W)$ of the QMaxSAT problem (line 6); or the hard part ϕ_H of the QBF is turned out to be false in itself. In the latter case QMSU1 returns false (line 8). By construction, $\mathcal{A}_S$ (if it exists) maximizes the number of satisfied clauses of φ_S and $\overrightarrow{Q} . \phi_H|_{\mathcal{A}_S}$ is true, i.e. it is the solution $\mathcal{A}_S^*$ of the QMaxSAT problem. Note that the algorithm is analogous to the MSU1 algorithm for non-quantified MaxSAT. The only difference is that QMSU1 uses not a SAT solver as an oracle but a QBF solver, and the hard part of the formula can be in a non-CNF form. Thus, the correctness of the algorithm relies on the corresponding result for the MSU1 algorithm [24].

Note that the only requirement imposed by the QMSU1 algorithm on the QBF oracle is the ability to produce a solution of a formula or its unsatisfiable core in case if it is true or false, respectively. While providing a solution of a formula seems straightforward to implement, the oracle must also be able to explain why the input formula is false, i.e. to extract an unsatisfiable core from the formula. A simple solution is to include all the soft clauses in the core. However, the efficiency of the algorithm relies on producing small cores.

4.2 Core extraction in QMaxSAT

While the QMSU1 algorithm can use any QBF solver as long as it produces unsatisfiable cores, for particular QMaxSAT applications (the smallest MUS problem and the smallest MES problem, see Section 5) this work uses a CEGAR-based 2QBF solver [38] as an underlying QBF oracle. Therefore, this section mainly describes a method for extracting unsatisfiable cores using a CEGAR-based QBF solver [37, 38]. Additionally, it also briefly mentions how this can be also done in QDPLL-based solvers.

4.2.1 Extracting cores in CEGAR-based QBF

Among the many practical uses of the *counterexample guided abstraction refinement* (CEGAR) [20], it can also be applied for solving QBF [37, 38]. The key idea of CEGAR is to consider an approximate representation of a problem (called the abstraction) instead

of its explicit representation that could be too large to construct or unknown. This section provides a basic overview of the algorithm for 2QBF (with the prefix $\exists X \forall Y$) and describes its modification, which is able to extract an unsatisfiable core of a formula if the formula is false. Additionally, it also extends the algorithm for the formulas with an arbitrary number of quantification levels. The reader is referred to [37, 38] for further details and properties of the algorithm.

For the sake of succinctness, when talking about 2QBF, assignments to variables of X and Y are denoted by μ and ν, respectively. We also assume, that the matrix of the 2QBF is presented as $\phi_H \wedge \phi_S$, where ϕ_S represents a set of soft clauses, and ϕ_H is a hard part given in a possibly non-CNF form. The algorithm hinges on the idea that the problem $\exists X \forall Y.\, \phi_H \wedge \phi_S$ can be equivalently represented as

$$\exists X. \bigwedge_{\nu \in \{0,1\}^{|Y|}} (\phi_H \wedge \phi_S)|_\nu \tag{4}$$

where the universal quantifier is *expanded* using a conjunction. Note that by construction (see Section 4.1.2), the set of soft clauses contains only variables existentially quantified on the outermost quantification level, i.e. variables of X. Therefore, instead of (4) one can equivalently consider

$$\exists X.\, \phi_S \wedge \bigwedge_{\nu \in \{0,1\}^{|Y|}} \phi_H|_\nu \tag{5}$$

Since the full expansion (5) of the problem can be exponentially large with respect to the original problem, it is infeasible to construct such representation in practice. Instead of constructing the full expansion (5), CEGAR constructs a *partial expansion* of the given problem, i.e.

$$\exists X.\, \phi_S \wedge \bigwedge_{\nu \in W} \phi_H|_\nu \tag{6}$$

where $W \subseteq \{0,1\}^{|Y|}$. We refer to formula (6) as W-abstraction. Observe that for any W, the corresponding W-abstraction is weaker than the full expansion (5). This means that the set of the W-abstraction's solutions is a superset over the set of solutions of the original problem, i.e. some of the W-abstraction's solutions may *not* satisfy (5). The idea of the CEGAR-based algorithm described below is to gradually strengthen the abstraction until a solution of the original problem is found, or the abstraction is proved to be false (see Fig. 1).

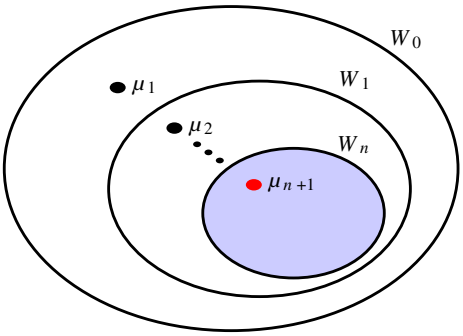


Fig. 1 Gradual strengthening of abstractions until a solution is found

Algorithm 3 CEGAR loop for 2QBF with core extraction

input : $\exists X \forall Y.\ \phi_H \wedge \phi_S$ s.t. ϕ_S is a set of clauses over X
output : $(\mathsf{true}, \emptyset, \mu)$ if there exists μ s.t. $\forall Y.\ (\phi_H \wedge \phi_S)|_\mu$, $(\mathsf{false}, \phi'_S, \emptyset)$ s.t. $\phi'_S \subseteq \phi_S$ otherwise

```
1  ω ← ∅
2  while true:
3      α ← ⋀_{ν∈ω} CNF(φ_H|_ν) ∧ φ_S
4      (st_1, φ_C, μ) ← SAT(α)                                  // candidate
5      if st_1 = false:
6          φ'_S ← φ_S ∩ φ_C
7          return (false, φ'_S, ∅)                              // no candidate found
8      (st_2, _, ν) ← SAT(¬(φ_H ∧ φ_S)|_μ)                      // counterexample
9      if st_2 = false:
10         return (true, ∅, μ)                                  // solution found
11     ω ← ω ∪ {ν}                                              // refine
```

Algorithm 3 shows the pseudo-code of the algorithm. The algorithm maintains a set of assignments W in the variable ω. We start with the abstraction equal to true, any assignment μ to the variables of X satisfies the abstraction. Assume, that the algorithm encodes the hard part ϕ_H into a CNF formula by calling a function $\mathsf{CNF}(\phi_H)$.

In each iteration of the loop, the algorithm first computes a solution to the abstraction, which is maintained in α and constructed at line 3. We refer to this solution as a *candidate solution*, because it is *not* guaranteed that it is indeed a solution to the original problem. If a SAT oracle says (see line 4) that there is no candidate solution, i.e. the abstraction has no solutions, the original problem does not have any solutions either (recall that the abstraction is always weaker than the problem). In this case the algorithm returns an unsatisfiable core of the input formula (line 7). Observe that the soft part ϕ'_S of the QBF core can be easily extracted from the core ϕ_C returned by the SAT oracle since it coincides with the soft part of ϕ_C, i.e. $\phi'_S = \phi_S \cap \phi_C$. In other words, the unsatisfiable core ϕ_C shows the falsity of the W-abstraction even if we consider the abstraction's soft part to be ϕ'_S instead of ϕ_S, i.e. $\exists X.\ \phi'_S \wedge \bigwedge_{\nu \in W} \phi_H|_\nu$ is false.

If the SAT oracle says that there is a candidate solution, then the algorithm checks whether it is indeed a solution of the problem or not. This is done by computing a *counterexample*. For a candidate μ, a counterexample ν is an assignment to the variables of Y such that $\neg(\phi_H \wedge \phi_S)|_\mu$. A counterexample ν serves as a witness that μ is not a solution, i.e. it is not the case that $\forall Y.\ \alpha|_\mu$ because $\phi_H \wedge \phi_S$ is false when variables of Y are assigned to ν. If no counterexample is found, the current candidate is indeed a solution and can be returned. If a counterexample *is* found, it is added to the set ω, which effectively strengthens the abstraction .

4.2.2 Extending RAReQS with core extraction

Following the ideas of [37], the method for core extraction described above can be extended to the case of formulas with an arbitrary number of quantification levels. Hence, we assume that we are given a false formula $\exists X \overrightarrow{Q}.\ \phi_H \wedge \phi_S$, where ϕ_S is in CNF and is defined only on the variables X. The objective is to compute a set $\phi'_S \subseteq \phi_S$ such that $\exists X \overrightarrow{Q}.\ \phi_H \wedge \phi'_S$ is also false. The algorithm RAReQS [37] is a recursive extension of the algorithm AReQS [38] (the core extraction for AReQS was described in Section 4.2.1). We will show that in order

to enhance RAReQS with core extraction, it is sufficient to modify only the top level call of recursion. Hence, we assume that the algorithm RAReQS is already implemented as described in [37]. RAReQS is represented by the function $\mathsf{RAReQS}(\Phi)$, about which we assume that for a given QBF $\Phi = \forall Y.\, \Psi$ it returns a pair $(\mathsf{st}, \mathcal{A}_Y)$ so that st is true iff Φ is true. The assignment $\mathcal{A}_Y$ is meaningful only if st is false, in which case $\mathcal{A}_Y$ is a complete assignment to Y such that $\Phi|_{\mathcal{A}_Y}$ is false.[8]

As in the case of two-level quantification, RAReQS partially expands quantifiers. Let us look at the case when we are solving a formula of the form $\exists X \forall U \exists Z.\, \Psi$. For simplicity, let's assume that U was expanded by two assignments μ_1 and μ_2, which yields the formula $\exists X.\, ((\exists Z.\, \Psi|_{\mu_1}) \wedge (\exists Z.\, \Psi|_{\mu_2}))$. In contrast to the 2QBF case, the expansion is no longer in prenex form. This can be fixed by introducing fresh variables for each copy of Z, hence obtaining: $\exists X Z^1 Z^2.\, \Psi^1|_{\mu_1} \wedge \Psi^2|_{\mu_2}$, where Ψ^i is obtained from Ψ by renaming variables Z to the corresponding variables Z^i. If the expansion is false, then the original formula is also false and we can stop. If the expansion is true, we obtain an assignment to the variables $X Z^1 Z^2$ that makes the expansion true. In such case we need to test that this assignment also makes true the input formula.

Algorithm 4 Extending RAReQS with core extraction

```
1  Function CoreRAReQS (∃X Q⃗. φ_H, φ_S)
   input  : ∃X. Q⃗. φ_H is a QBF in prenex closed form; φ_S is a set of clauses over X
   output : (true, ∅, A_X) if there exists A_X s.t. Q⃗. (φ_H ∧ φ_S)|_{A_X} is true.
            (false, φ'_S, ∅) s.t. φ'_S ⊆ φ_S otherwise
2  begin
3      if Q⃗ has no quantifiers:
4          (st, φ_C, μ) ← SAT(CNF(φ_H) ∧ φ_S)
5          return (st, φ_S ∩ φ_C, μ)                          // extract a core if any
6      ω ← ∅
7      while true:
8          α ← ⋀_{ν∈ω} (Q⃗. φ_H)|_ν                            // build an abstraction
9          (st_1, φ'_S, μ') ← CoreRAReQS(Prenex(∃X. α), φ_S)    // find a candidate
10         if st_1 = false:
11             return (false, φ'_S, ∅)                        // abstraction false
12         μ ← {l | l ∈ μ' ∧ var(l) ∈ X}                      // filter a candidate
13         (st_2, _, ν) ← RAReQS(Q⃗. φ_H|_μ)                   // find a counterexample
14         if st_2 = true:
15             return (true, ∅, μ)                            // solution found
16         ω ← ω ∪ {ν}                                        // refine
17 end
```

Algorithm 4 concretizes this general idea. The algorithm is represented by the recursive function $\mathsf{RAReQS}(\exists X \overrightarrow{Q}.\, \phi_H, \phi_S)$, which expects the input formula to be specified as a pair consisting of the hard part of the formula and the soft part. Semantically, the input corresponds to the QBF $\exists X \overrightarrow{Q}.\, \phi_H \wedge \phi_S$, but it will be useful for us to keep the soft clauses separately. The function returns a triple $(\mathsf{st}, \phi'_S, \mathcal{A}_X)$ so that st is true if and only if the input formula is true. If st is true, then $\mathcal{A}_X$ is a complete assignment to the X variables such that $\overrightarrow{Q}.\, (\phi_H \wedge \phi_S)|_{\mathcal{A}_X}$ is true. If st is false, then ϕ'_S is a core, i.e. $\exists X \overrightarrow{Q}.\, \phi_H \wedge \phi'_S$ is also false.

[8] In fact, we could use any other QBF solver satisfying this interface.

If $\overrightarrow{Q}$ contains no quantifiers, this is the base case of the recursion as we can simply invoke a SAT solver and use its capacity to produce either a core or satisfying assignment (lines 4–5). Similarly to Algorithm 3, the algorithm is assumed to be able to encode the hard part ϕ_H into CNF by calling $\mathsf{CNF}(\phi_H)$.

In the non-base case, the algorithm maintains in the variable ω the assignments used for the expansion—initialized to be empty. The abstraction is constructed by expanding the first quantifier of $\overrightarrow{Q}$, which is necessarily the universal quantifier, and therefore, is expanded as a conjunction (in the worst-case, this expansion may lead to an exponentially large formula, corresponding to the exact semantics of the universal quantifier).

Semantically, the abstraction corresponds to $\bigwedge_{\nu\in\omega}(\overrightarrow{Q}.\phi_H \wedge \phi_S)|_\nu$. However, since ϕ_S is defined only on the variables X, it is not affected by the expansion and therefore it can be taken out, yielding the formula $\phi_S \wedge \bigwedge_{\nu\in\omega}(\overrightarrow{Q}.\phi_H)|_\nu$. This formula is prenexed (by introducing new variables) and is solved by a recursive call (line 9).

The recursive call tells us either that the abstraction is false and then we stop, or, if the abstraction is true, we test whether the assignment satisfying the abstraction also satisfies the input formula. If the abstraction is false, we use the core of the abstraction as the core of our formula (line 11). If the abstraction is true, the recursive call gives us an assignment to the variables X but also to the auxiliary variables that were introduced by prenexing; these are filtered out as they are not of interest, giving us the candidate μ (line 12).

To test that the candidate makes our original formula true, we invoke the function RAReQS on $\overrightarrow{Q}.\phi_H|_\mu$. Note that at this stage ϕ_S can be safely omitted because it is defined only on the X variables, and, it must have been necessarily satisfied by the candidate, i.e. $\phi_S|_\mu = 1$. If the call to RAReQS tells us that reducing by μ yields a true formula, we are done (the original formula is true). If, the reduced formula is false, we use the assignment returned by RAReQS for refinement (line 16).

Core extraction for DPLL-based QBF solvers Extracting cores in the DPLL-based QBF solvers [73] is quite similar to the extraction of cores in SAT solvers [25, 74]. Essentially, there are two approaches to extracting cores: from a *refutation proof* or by the use of *assumptions*. If a QBF solver traces its workings, it can produce a Q-resolution refutation [40, 57, 70][9] of the formula and the leaves of such proof serve as the core. In the assumption method, we require a QBF solver to accept a set of literals that must be true (assumptions). Further, if the formula is false, it returns a subset of these assumptions that are responsible for the formula being false [48]. Then, every soft clause c is adorned with a fresh literal $\neg s_c \vee c$, and the solver is called with the assumptions $\{s_c | c \in \phi_S\}$. Like so the assumption s_c insists the clause c be satisfied. If the formula is false (unsatisfiable), the solver gives us a subset of the assumptions that correspond to the core.

5 Applications of QMaxSAT

This section describes two applications of the proposed quantified MaxSAT framework. Both applications consist in minimizing a set of clauses with respect to unsatisfiability or equality criteria. More precisely, we are going to present the SMUS and SMES problems. In contrast to finding a smallest possible subset of clauses that are unsatisfiable or equivalent to

[9] Some solvers require *long-distance Q-resolution* [9, 23, 73] and further extensions might be needed if preprocessing is applied [10, 32, 36, 69].

the original CNF formula, the MUS and MES problems are well studied and the algorithms for solving them are deployed in practice [11, 13, 30, 49, 63, 65, 67].

Although in many practical applications minimization of an unsatisfiable/ equivalent set of clauses can be quite expensive, the resulting subset can be relatively close to the exact optimum (i.e. a smallest MUS/MES) and, thus, satisfactory. However, in some cases there is no guarantee for the state-of-the-art MUS/MES extracting algorithms to obtain an MUS (or MES) of a reasonable size (i.e. close to the exact optimum). The following example gives an intuition why this is indeed true. Let us consider a CNF formula $\varphi = \{c_1 = x_1, c_2 = \neg x_1 \vee x_2, c_3 = \neg x_1 \vee x_3, c_4 = \neg x_1 \vee x_4, c_5 = \neg x_1 \vee x_5, c_6 = \neg x_2 \vee \neg x_3 \vee \neg x_4 \vee \neg x_5, c_7 = x_6 \vee x_7, c_8 = \neg x_6 \vee \neg x_7, c_9 = x_6 \vee \neg x_7, c_{10} = \neg x_6 \vee x_7\}$. Observe that formula φ has exactly two MUSes: $\varphi^1 = \{c_1, c_2, c_3, c_4, c_5, c_6\}$ and $\varphi^2 = \{c_7, c_8, c_9, c_{10}\}$. Normally, a modern MUS extractor would find and report φ^1 while the smallest MUS is φ^2. This can be caused by various factors: clause ordering, heuristics used in MUS/MES extractors (e.g. *trimming*), and/or even the implementation of the SAT oracle (more precisely by the use of *unit propagation*). In practice the size of the MUSes/MESes can vary a lot. Therefore, if a user prefers smaller MUSes/MESes, it might be crucial to use specific algorithms for extracting the smallest MUS/MES instead of relying on the result produced by an MUS/MES extractor.

5.1 The smallest MUS problem

This section considers a concrete application of the QMaxSAT problem — the problem of finding a smallest MUS of a CNF formula. Let $X = \{x_1, \ldots, x_n\}$ be a set of Boolean variables and $\varphi = \{c_1, \ldots, c_m\}$ be a CNF formula. Formula $\varphi_{\text{mus}} \subseteq \varphi$ is called a *minimal unsatisfiable subformula* (MUS) of φ, if φ_{mus} is unsatisfiable and $\forall c_i \in \varphi_{\text{mus}}$ formula $\varphi_{\text{mus}} \setminus \{c_i\}$ is satisfiable. The MUS problem is a subject of active research (e.g. [51]).

Definition 3 Formula $\varphi^*_{\text{mus}}, \varphi^*_{\text{mus}} \subseteq \varphi$, is called a smallest MUS of φ if

1. φ^*_{mus} is unsatisfiable;
2. any subformula $\varphi' \subseteq \varphi$ s.t. $|\varphi'| < |\varphi^*_{\text{mus}}|$ is satisfiable.

The *smallest MUS* problem (SMUS) consists in finding a smallest MUS of a CNF formula. An algorithm that computes an SMUS by searching the space of all unsatisfiable subformulas was presented in [49]. A greedy genetic algorithm that finds approximate solutions of the SMUS problem was proposed in [71]. A branch and bound algorithm for computing SMUSes was described in [45, 53]. The decision version of the SMUS problem, i.e. the problem of determining whether a given formula has a smallest MUS of size k, is known to be Σ_2^P-complete (e.g. see [29]).

Let us formulate an optimization extension of SMUS in terms of QMaxSAT defined in Section 3. First, we introduce a set of *selection variables* $\{s_1, \ldots, s_m\}$ denoted by S. Then we extend each clause $c_i \in \varphi$ with a selection literal $\neg s_i$. Let us denote the resulting CNF formula by $\varphi_+ = \{c_1 \vee \neg s_1, \ldots, c_m \vee \neg s_m\}$ (subscript "+" means that it is an extended φ). Now instead of the original CNF formula φ, we construct two formulas: a non-CNF hard part $\phi_H = \neg\varphi_+$, and a set of soft clauses $\phi_S = \{\neg s_1, \ldots, \neg s_m\}$. Note the soft part ϕ_S is different from the one considered in Section 3 since the soft clauses contain negative literals $\neg s_i$. The purpose of this will become clear below.

Let us consider a QBF $\psi_{\text{unsat}} = \exists S \forall X.\ \phi_H$. This QBF is true, i.e. $\mathcal{M}(\psi) \neq \emptyset$, iff the original CNF formula φ has an unsatisfiable subformula. Also notice that if all selection variables are assigned false (none of the clauses are selected), the hard part ϕ_H is false

since an empty CNF cannot have an unsatisfiable subformula. Now we can construct a QBF formula, which the QMaxSAT formulation of the SMUS problem will be done for. This formula a simplified version of (2) and is the following:

$$\psi = \exists S \, \forall X. \, \phi_H \wedge \phi_S \tag{7}$$

This formula is false and the problem of finding a smallest MUS of φ consists in finding an assignment $\mathcal{A}_S^*$ that defines a subset of soft clauses ϕ_S^*, $\phi_S^* \subseteq \phi_S$, of the largest possible size such that QBF $\forall X. \, \phi_H \wedge \phi_S^*$ is true. The semantics of ϕ_S^* for the original CNF φ is that all the subsets of φ that have a smaller size than ϕ_S^* do not have an unsatisfiable subformula and, hence, ϕ_S^* defines the smallest unsatisfiable subformula of the original CNF formula φ.

As shown in Section 4.1.1, to solve this problem, one can use an iterative approach calling a 2QBF oracle, which decides whether the following quantified formula is true or false: $\exists S \, \forall X. \, \phi_H \wedge (\sum_{s \in S} s \leq k)$. The solution cost k changes with each iteration. However, one can also apply algorithm QMSU1 to this problem as well, which will iteratively ask the QBF oracle to decide QBF (7).

The QMSU1 algorithm iteratively extracts unsatisfiable cores of formula (7) and relaxes their soft parts, which are some subsets of ϕ_S, until it finds an assignment $\mathcal{A}_S^*$ that maximizes the number of satisfied clauses of ϕ_S. Assignment $\mathcal{A}_S^*$ defines the largest subset ϕ_S^* of satisfied clauses of ϕ_S. Subset ϕ_S^* in turn corresponds to an SMUS φ_{mus}^*, $\varphi_{\text{mus}}^* \subseteq \varphi$, such that a clause $c_i \in \varphi_{\text{mus}}^*$ iff $\mathcal{A}_S^*(s_i) = 1$.

5.2 The smallest MES problem

Another example of a concrete practical application of the QMaxSAT problem is the problem of computing a *smallest minimal equivalent subformula* (*smallest MES*, or *SMES*) of a CNF formula.

Definition 4 Given a CNF formula φ, formula φ_{mes}, $\varphi_{\text{mes}} \subseteq \varphi$, is called a minimal equivalent subformula of φ if $\varphi_{\text{mes}} \equiv \varphi$ and $\forall_{c \in \varphi_{\text{mes}}}$, $\varphi_{\text{mes}} \setminus \{c\} \not\equiv \varphi$.

The problem of computing MESes of a CNF formula [11] is tightly connected to the notions of a *redundant clause* and *irredundant formula*. A clause $c \in \varphi$ is said to be redundant if $\varphi \setminus \{c\} \models c$. In other words, adding/removing c to/from φ does not change the set of models of φ. Formula φ is referred to as *irredundant* if none of its clauses is redundant. Given a CNF formula, its irredundant subformulas are referred to as *irredundant cores* [43], or *irredundant equivalent subsets* [44]. It is not hard to see that an MES of a CNF formula is irredundant. It was shown in [44] that deciding whether a given CNF formula is irredundant (i.e. it is an MES) is D^P-complete.

Observe that the problem of computing an MES of CNF formula is defined for both satisfiable and unsatisfiable formulas while computing an MUS makes sense only if the formula is unsatisfiable. Moreover, given an unsatisfiable formula φ, an MES of φ is clearly one of its MUSes. Therefore, computing an MUS of a formula can be considered as a particular case of computing an MES. Analogously to SMUS, a smallest MES can be defined in the following way.

Definition 5 Formula φ_{mes}^*, $\varphi_{\text{mes}}^* \subseteq \varphi$, is called a smallest MES of φ if

1. $\varphi_{\text{mes}}^* \equiv \varphi$;
2. for any subformula $\varphi' \subseteq \varphi$ s.t. $|\varphi'| < |\varphi_{\text{mus}}^*|$ the following holds $\varphi' \not\equiv \varphi$.

Due to the close relationship between the SMUS and SMES problems, the latter can be represented in terms of QMaxSAT defined in Section 3 in a way similar to the one from Section 5.1. Namely, we introduce a set of selection variables $S = \{s_1, \ldots, s_m\}$. Given a CNF formula φ, we again construct a formula $\varphi_+ = \{c_1 \vee \neg s_1, \ldots, c_m \vee \neg s_m\}$, $c_i \in \varphi$. Obviously, assignments to variables of S define subsets of formula φ: a clause $c_i \in \varphi$ is *selected* (or *activated*) if the corresponding selection variable $s_i \in S$ is assigned true.

Next, we construct a QBF $\psi_{\text{equiv}} = \exists S \forall X.\ \varphi_+ \equiv \varphi$, which is true if formula φ has an equivalent subformula. Observe that equivalence $\varphi \equiv \varphi_+$, by definition, means $(\varphi \models \varphi_+) \wedge (\varphi_+ \models \varphi)$. Clearly, the first entailment holds by construction (as mentioned above, given an assignment to the selection variables, φ_+ represents a subset of φ). And the second entailment can be represented by $\phi_H = \neg\varphi_+ \vee \varphi$. Thus, QBF ψ_{equiv} can be equivalently represented as $\exists S \forall X.\ \phi_H$.[10]

Considering a set of soft clauses from Section 5.1, i.e. $\phi_S = \{\neg s_1, \ldots, \neg s_m\}$, enables to formulate the SMES problem in terms of QMaxSAT: find an assignment $\mathcal{A}^*_S$ that defines a subset of soft clauses ϕ^*_S, $\phi^*_S \subseteq \phi_S$, of the largest possible size such that QBF $\forall X.\ \phi_H \wedge \phi^*_S$ is true.

Again, there are several approaches to this problem. One is to iteratively (by incrementing or decrementing k, see Section 4.1.1) decide the following quantified formula $\exists S \forall X.\ \phi_H \wedge (\sum_{s \in S} s \leq k)$.

However, in this work we mainly focus on applying the core-guided algorithms proposed in Section 4.1.2, which call a QBF oracle to decide whether QBF $\exists S \forall X.\ \phi_H \wedge \phi_S$ is true of false. Initially this formula is false, and the core-guided algorithms proposed in Section 4.1.2 iteratively extract its unsatisfiable cores and relax their soft parts until the formula becomes true.

5.3 Problem specific lower bounds

Most of the state-of-the-art MaxSAT solvers use heuristics to improve the performance. For example, in practice, it is quite important for a MaxSAT solver to compute good lower and upper bounds on the optimal value so that the solver could skip unnecessary iterations. Similar ideas can be used for improving the performance of the quantified MaxSAT algorithms proposed in Section 4.1. However, it should be noted that in contrast to classical MaxSAT, where the formula is always in CNF, for the case of QMaxSAT matrices of QBFs can be specified in a potentially non-CNF form. Therefore, heuristics for computing lower and upper bounds can vary significantly depending of the formula type. Here we briefly describe how one can compute a lower bound for the QMaxSAT formulation of the two problems described above: SMUS and SMES.

Let us describe the idea for the case of the SMUS problem.[11] It was originally proposed and applied to the Digger algorithm. To increase its performance, Liffiton et al. [45] use a preprocessing technique — computing a set of *disjoint MCSes*. An MCS (or *minimal correction set*) of an unsatisfiable CNF formula φ is a subset of clauses $\delta \subset \varphi$ such that $\varphi \setminus \delta$ is satisfiable while $\varphi \setminus \delta \cup c$ is unsatisfiable for any clause $c \in \delta$. The heuristic used in Digger hinges on the following important connection between MCSes and MUSes of a CNF formula formulas (see [7, 17, 30, 46, 47, 61]): any MUS of formula φ is a *minimal*

[10] One can easily see that in the case when φ is unsatisfiable, formula ψ_{equiv} degenerates to ψ_{unsat} from Section 5.1. This confirms that the MES problem is a generalization of the MUS problem.

[11] However, since the SMES problem is a generalization of SMUS, similar reasoning can be applied for computing a lower bound for SMES. The only difference is the formula to consider (see Section 5.2 for details).

hitting set of the complete set of MCSes of φ. Therefore, the enumeration of disjoint MCSes gives a lower bound of the size of an SMUS, thus, reducing the search space of the Digger algorithm.

In our QMaxSAT approach, we can exploit the same technique only for iterative algorithms (i.e. QLSUS, QLSSU, and QBINS) since the QMSU1 algorithm does not handle constraints $\leq k$ directly. Thus, it is not enough for QMSU1 to have a lower bound. In order to use it, QMSU1 needs some cores that witness the lower bound to be correct. However, the enumeration of disjoint MCSes can be still helpful while solving SMUS by QMSU1. For example, if a CNF formula φ has an MCS $C = \{c\}$, where c is a clause (so called *unit* MCS), then each MUS of φ contains clause c (due to the minimal hitting set duality). Therefore, one of the improvements of QMSU1 for computing an SMUS of formula φ can be enumeration of all the unit MCSes of φ.[12] This makes it easier to solve the SMUS/SMES problem by pruning the search space extensively since the solution of the problem must include all the clauses comprising the unit MCSes enumerated.

Although the enumeration of unit MCSes helps to improve the QMSU1 algorithm, in practice it is often not enough to consider just unit MCSes. Therefore, another technique we exploit in our approach is the use of disjoint MCSes, found during the preprocessing stage, as unsatisfiable cores of formula (7). Let δ be an MCS of φ. By φ_S^δ we denote a subformula of φ_S containing only clauses of φ_S that correspond to clauses of δ, i.e. $(\neg s_i) \in \varphi_S^\delta$ if $c_i \in \delta$. By definition of an MCS, formula $\varphi \setminus \delta$ is satisfiable. This means that $\varphi_R \wedge \varphi_S^\delta$ is also satisfiable. Then the formula $\exists S \forall X.\ \neg\varphi_R \wedge \varphi_S^\delta$ is false. Given Definition 2, this means that $\neg\varphi_R \wedge \varphi_S^\delta$ is a core of (7). Therefore, k MCSes computed by preprocessing give us k unsatisfiable cores of (7). Moreover, since all the computed MCSes are disjoint, the cores are disjoint. So the use of this preprocessing technique provides the algorithm with a lower bound and a set of disjoint unsatisfiable cores that are the reason for the lower bound. Following lines 10–13 of Algorithm 2, all of these cores can be then processed before running the algorithm itself. In practice this significantly increases the performance of the QMSU1 algorithm (see Section 6).

Algorithm 5 MaxSAT-based algorithm for enumerating disjoint MCSes

input : Unsatisfiable CNF formula φ, s.t. $\mathcal{M}(\varphi) = \emptyset$
output : Set of pairwise disjoint MCSes $\Delta(\varphi)$

```
1  Δ(φ) ← ∅                                          // initially Δ(φ) is empty
2  while SAT(Δ(φ)) = true:
3      φ' ← MaxSAT(φ)                                // running a MaxSAT solver for φ
4      while true:                                   // enumerating MCSes based on models of φ'
5          (st, A) ← SAT(φ')
6          if st = false:
7              break
8          δ ← {c ∈ φ, c|_A = false}                 // δ includes every clause c ∈ φ falsified by A
9          φ' ← φ' ∪ δ
10         Δ(φ) ← Δ(φ) ∪ δ
11     φ ← φ'
```

[12] Unit MCSes are about 60–80 % of the total number of disjoint MCSes computed for the benchmarks considered in the experimental evaluation (see Section 6).

There is a number of efficient algorithms for enumerating MCSes of a CNF formula (e.g. see [52, 58] and references therein). Algorithm 5 shows a pseudo-code of a typical MaxSAT-based algorithm for computing disjoint MCSes of an unsatisfiable CNF formula used as a preprocessing heuristic in our QMaxSAT approach. Given an unsatisfiable CNF formula φ as an input, the algorithm constructs a set of pairwise disjoint MCSes of φ. Algorithm 5 computes MCSes of φ starting from the the smallest ones (in terms of the number of clauses). In order to do so, it calls a MaxSAT solver for formula φ (see line 3). Note that while solving the MaxSAT problem for φ, the MaxSAT solver does a series of relaxations of formula's unsatisfiable cores and constructs the corresponding cardinality constraints. When the solver terminates, the resulting modified formula φ' is obtained, which is satisfiable. Each model $\mathcal{A}$ of φ' (line 5) defines a MaxSAT solution for the original formula φ. As a side effect, it also defines an MCS δ corresponding to that MaxSAT solution, i.e. MCS δ comprises clauses of φ that are falsified by $\mathcal{A}$ (line 8). Each MCS δ found by this process is then blocked (see lines 9–10) so that next model of φ' must satisfy all of its clauses. Algorithm 5 iterates while the set $\Delta(\varphi)$ of all disjoint MCSes of φ is satisfiable. Note that satisfiability of formula $\Delta(\varphi)$ means that there is a *maximally satisfiable* set of clauses $\mu \subset \varphi$ s.t. $\Delta(\varphi) \subseteq \mu$ implying that there is an MCS $\delta = \varphi \setminus \mu$ such that $\delta \cap \Delta(\varphi) = \emptyset$ (and, thus, MCS δ can be still identified and added to $\Delta(\varphi)$). Unsatisfiability of $\Delta(\varphi)$ means that all disjoint MCSes of φ are already included into $\Delta(\varphi)$.

6 Experimental results

6.1 The SMUS problem

A prototype of a solver for the SMUS problem implementing the QMSU1 algorithm was developed with the use of a CEGAR-based 2QBF oracle described in Section 4.2.1. The underlying SAT solver of our 2QBF oracle implementation is MINISAT 2.2 [22]. We refer to this prototype as *MinUC* (**Min**imum **U**nsatisfiable **C**ore finder). Three versions of this solver were developed. The default one is the core-guided version. The other two include *MinUC-LB* and *MinUC-UB* and implement iterative linear lower and upper bound approaches respectively. In order to do a comprehensive comparison between QMSU1 and Digger, we ran MinUC in three different modes (the corresponding names of the tools are presented in the parentheses):

- without enumerating disjoint MCSes (*MinUC-w*);
- with the use of the Digger's disjoint MCS enumerator (*MinUC-d*);
- with the use of the default built-in[13] disjoint MCS enumerator (*MinUC*).

The set of instances considered includes several sets of benchmarks described below. The first set consists of automotive product configuration benchmarks [65]. Two other sets of benchmarks come from circuit diagnosis. Additionally, we selected instances from the complete set of the MUS competitions benchmarks[14] as follows. Since the SMUS problem is computationally harder than the problem of extracting an MUS of formula, we picked all

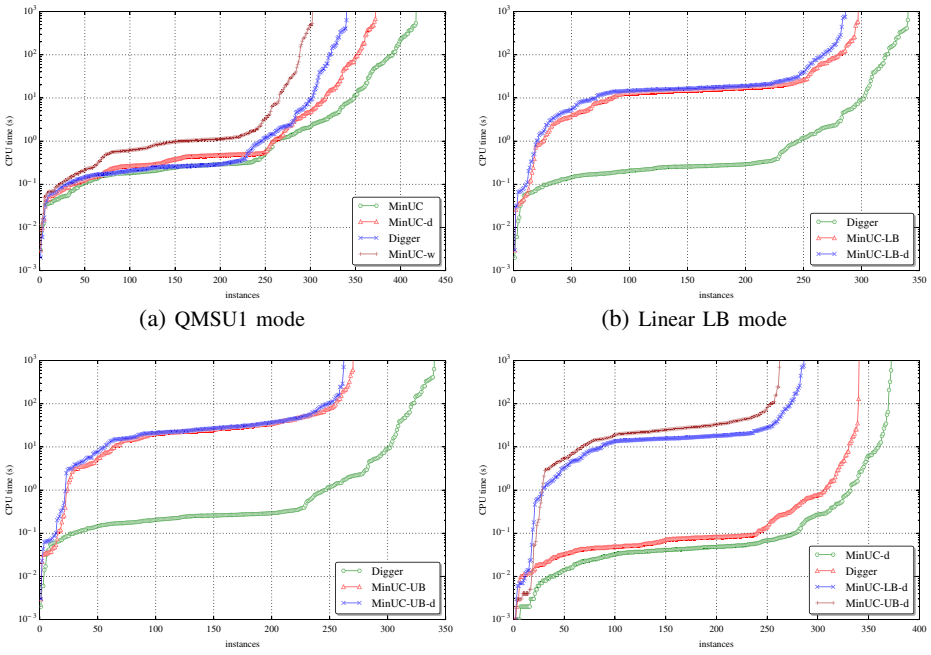
[13] The algorithm used as the default disjoint MCS enumerator of MinUC is MaxSAT-based and described in Section 5.3.

[14] http://www.satcompetition.org/2011/

the instances from the MUS competitions that can be solved by MUSER-2 (see [13]) in less than 10 seconds. The total number of instances in the set is 682. All experimental results were obtained on an Intel Xeon 5160 3GHz, with 4GB of memory, and running Fedora Linux operating system. The experiments were made with a 800 second time limit and a 2GB memory limit. The detailed overview of the results is presented in the following plots.

Figure 2a shows a cactus plot illustrating the performance of the core-guided version of MinUC compared to Digger. The version of MinUC without enumerating disjoint MCSes (MinUC-w) can solve 325 instances only. Digger solves 364 instances while MinUC with the same MCS enumerator (MinUC-d) is able to solve 396 instances. This is 8.8 % more than by Digger's result (4.7 % of all the 682 instances). In the case of using its own MCS enumerator MinUC demonstrates the best performance with 444 instances solved, having 22 % advantage over Digger (11.7 % of the total 682 instances). Figure 2b and c show similar plots for linear search LB and UB modes respectively. Even with the use of its own MCS enumerator, linear search modes of MinUC perform worse than Digger: MinUC-LB solves 322 while MinUC-UB solves 294 instances. Figure 2d gives a more graphic comparison between Digger and all the versions of MinUC using Digger's MCS enumerator. In this case, the time required to enumerate disjoint MCSes is not taken into account (because it is the same for all the solvers) while in all the other cases it is included in the runtime.

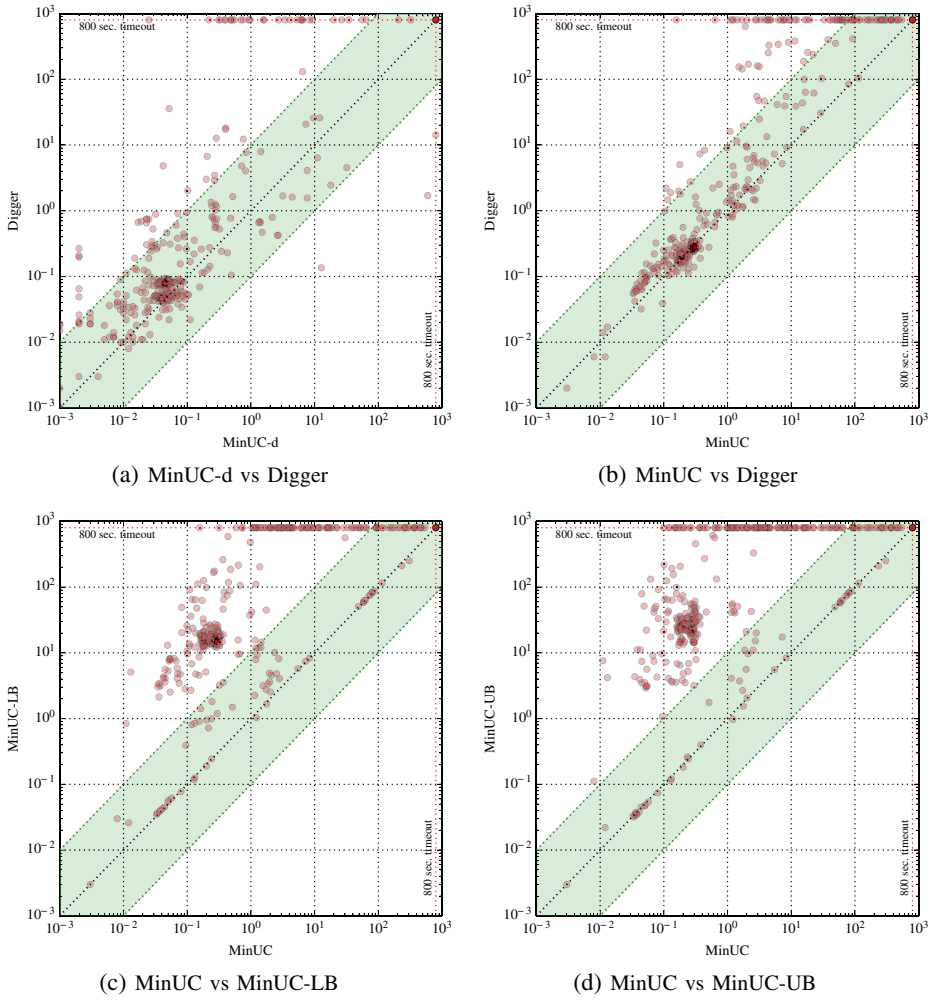
Figure 3 shows scatter plots comparing the QMSU1 versions of MinUC to Digger (see Fig. 3a and b) and to its linear search versions (Fig. 3c and d). Figure 4 gives an overview on how many instances are solvable either by Digger or by core-guided MinUC only.



(a) QMSU1 mode (b) Linear LB mode

(c) Linear UB mode (d) All modes with Digger's MCS enumerator

Fig. 2 Performance of MinUC compared to Digger



(a) MinUC-d vs Digger

(b) MinUC vs Digger

(c) MinUC vs MinUC-LB

(d) MinUC vs MinUC-UB

Fig. 3 Performance of the used approaches

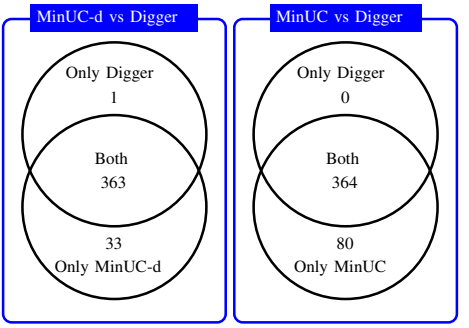


Fig. 4 Number of instances solvable by different solvers: Digger vs MinUC

The results indicate that the core-guided version of MinUC has an advantage over other approaches. Digger comes second. MinUC-LB and MinUC-UB have the worst performance. Although the experiment results are quite positive for the current version of the core-guided version of MinUC comparing to Digger, it is still has a potential for possible improvements.

6.2 The SMES problem

Besides evaluating quantified MaxSAT algorithms applied to the SMUS problem, we also implemented and ran them in order to solve the SMES problem described in Section 5.2. However, it should be noted that to the best of our knowledge, there are no algorithms nor tools targeted specifically on solving SMES. Therefore, we only compared performance of the three algorithms described in Section 4.1: QLSUS, QLSSU, and QMSU1. In order to solve the SMES problem, we implemented these algorithms in a prototype of a solver for SMES. The prototype is referred to as *SIEqS* (**S**mallest **I**rredundant **Eq**uivalent **S**ubformula finder) and uses MINISAT 2.2 [22] as the underlying SAT solver. In the experimental evaluation, acronyms *SIEqS-LB*, *SIEqS-UB*, and *SIEqS-MSU1* correspond to algorithms QLSUS, QLSSU, and QMSU1, respectively.

The benchmark set we chose for this evaluation includes instances considered in [11] that come from practical application domains of SAT including planning problem instances. Since finding a smallest MES is computationally much harder than finding any MES, we selected only those instances that are relatively easy for an MES extractor. More precisely, the instances were filtered by running MUSER-2 in the MES mode for 10 seconds. The total number of instances in the considered set is 164.

Figure 5 shows the cactus plot comparing the considered algorithms. More detailed performance comparison of SIEqS to SIEqS-LB and SIEqS- UB is presented in Fig. 6a and b, respectively. Given the presented results, it is important to note the relevance of the unsatisfiable core extraction for the QMaxSAT algorithms applied to the SMES problem.

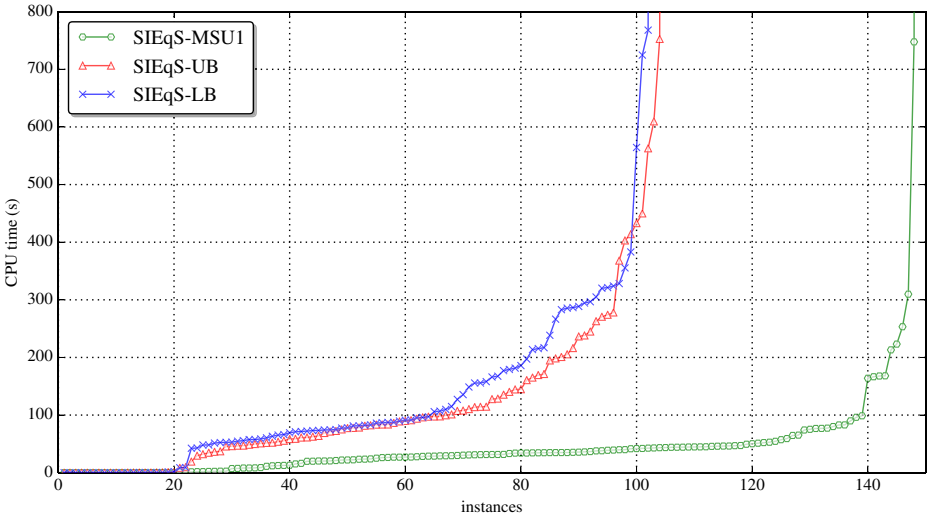
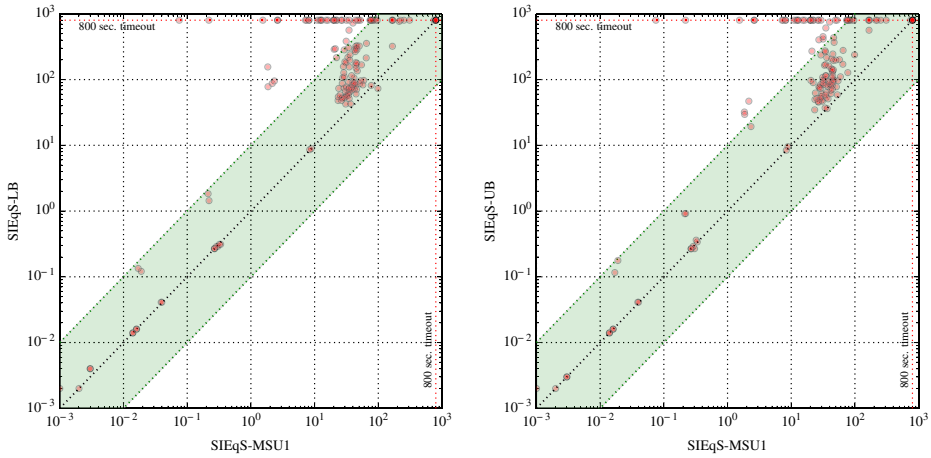


Fig. 5 Performance of SIEqS applied to SMES



(a) SIEqS-MSU1 vs SIEqS-LB (b) SIEqS-MSU1 vs SIEqS-UB

Fig. 6 Performance of QMSU1, QLSUS, and QLSSU applied to SMES

7 Conclusions

This paper studies optimization problems over quantified sets of constraints, and focuses on the concrete case of quantified MaxSAT (QMaxSAT). The main contributions of the paper are: (i) a novel core-guided algorithm for QMaxSAT; (ii) generation of unsatisfiable cores with CEGAR-based QBF solvers; (iii) a QMaxSAT-based approach for solving the smallest MUS (SMUS) and smallest MES (SMES) problems; and (iv) new pruning techniques for solving the SMUS and SMES problems. The novel core-guided algorithm for QMaxSAT is based on the original work of Fu & Malik [24]. Nevertheless, other algorithms for non-quantified MaxSAT can also be adapted to the quantified case (e.g. [4, 31]).

Experimental results on representative problem instances demonstrate that the novel approach for computing SMUSes, based on core-guided QMaxSAT algorithms, significantly outperforms Digger, a state-of-the-art algorithm for computing an SMUS. These results, as well as results for the SMES problem, motivate applying core-guided QMaxSAT algorithms to other optimization problems with quantified constraints.

A number of future research directions can be envisioned. Investigating additional optimization problems with quantified constraints will provide a larger set of problem instances. Motivated by a larger universe of problems and problem instances, additional core-guided algorithms can be developed for QMaxSAT. Finally, it will be important to investigate how to extend the algorithms developed in this paper to settings more general than QMaxSAT. For example, MAX-QSAT [21] among others [15, 19]. Any QBF solver can be integrated into the QMSU1 algorithm as an oracle as long as it produces unsatisfiable cores. There are known techniques for extracting unsatisfiable cores from unsatisfiable QBF instances for DPLL-based QBF solvers. One of these techniques is proposed in [70] and then followed by recent works on certificate generation for resolution-based QBF solvers (e.g. [8, 9, 48, 57]) and preprocessors [32, 36]. Thus, an interesting subject of future work is integration of a DPLL-based QBF solver into the QMSU1 algorithm and comparison of its performance (in terms of speed and a core size) with performance of the currently implemented

CEGAR-based core-producing QBF oracle. Additionally, it is known that QBF preprocessing improves the performance of QBF solvers extensively. Hence, another line of future work will be related to the possibility of applying QBF preprocessing to the QMaxSAT algorithms. Following the ideas of [14], it is interesting to determine what kinds of QBF preprocessing techniques can be applied to the QMaxSAT algorithms.

For the concrete applications of QMaxSAT, the SMUS and SMES problems, several optimizations can be considered. Modern (and efficient) MUS/MES solvers [11, 13] can be used for computing an upper bound on the size of the SMUS/SMES. If the lower bound (e.g. due to disjoint cores, or by iterative core extraction) equals the upper bound, then an SMUS/SMES will by given by any minimal hitting set of all the disjoint MCSes. Moreover, several preprocessing approaches can be used, several of which are more efficient than the one used in Digger.

Acknowledgments This work is partially supported by SFI PI grant BEACON (09/IN.1/ I2618), FCT grants ATTEST (CMU-PT/ELE/0009/2009) and POLARIS (PTDC/EIA-CCO/123051/2010), and national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013.

References

1. Andraus, Z.S., Liffiton, M.H., & Sakallah, K.A. (2006). Refinement strategies for verification methods based on datapath abstraction. In *ASP-DAC* (pp. 19–24). doi:10.1145/1118299.1118306.
2. Andraus, Z.S., Liffiton, M.H., & Sakallah, K.A. (2008). Reveal: a formal verification tool for verilog designs. In *LPAR* (pp. 343–352). doi:10.1007/978-3-540-89439-1_25.
3. Ansótegui, C., Bonet, M.L., & Levy, J. (2009). Solving (weighted) partial MaxSAT through satisfiability testing. In *SAT* (pp. 427–440).
4. Ansótegui, C., Bonet, M.L., & Levy, J. (2010). A new algorithm for weighted partial MaxSAT. In *AAAI*.
5. Ansótegui, C., Bonet, M.L., & Levy, J. (2013). SAT-based MaxSAT algorithms. *Artificial Intelligence*, *196*, 77–105.
6. Ansótegui, C., Gomes, C.P., & Selman, B. (2005). The Achilles' heel of QBF. In *AAAI* (pp. 275–281).
7. Bailey, J., & Stuckey, P.J. (2005). Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization. In *PADL* (pp. 174–186).
8. Balabanov, V., & Jiang, J.H.R. (2011). Resolution proofs and Skolem functions in QBF evaluation and applications. In G. Gopalakrishnan, S. Qadeer (Eds.), *CAV* (pp. 149–164).
9. Balabanov, V., & Jiang, J.H.R. (2012). Unified QBF certification and its applications. *Formal Methods in System Design*, *41*(1), 45–65.
10. Balabanov, V., Widl, M., & Jiang, J.H.R. (2014). QBF resolution systems and their proof complexities. In C. Sinz, U. Egly (Eds.), *SAT* (pp. 154–169). Springer.
11. Belov, A., Janota, M., Lynce, I., & Marques-Silva, J. (2012). On computing minimal equivalent subformulas. In M. Milano (Ed.), *CP* (Vol. 7514, pp. 158–174). Springer.
12. Belov, A., Janota, M., Lynce, I., & Marques-Silva, J. (2014). Algorithms for computing minimal equivalent subformulas. *Artificial Intelligence*, *216*, 309–326. doi:10.1016/j.artint.2014.07.011.
13. Belov, A., Lynce, I., & Marques-Silva, J. (2012). Towards efficient MUS extraction. *AI Communication*, *25*(2), 97–116.
14. Belov, A., Morgado, A., & Marques-Silva, J. (2013). SAT-based preprocessing for MaxSAT. In *LPAR* (pp. 96–111). doi:10.1007/978-3-642-45221-5_7.
15. Benedetti, M., Lallouet, A., & Vautard, J. (2008). Quantified constraint optimization. In *CP* (pp. 463–477).
16. Berre, D.L., & Parrain, A. (2010). The Sat4j library, release 2.2. *JSAT*, *7*(2-3), 59–6.
17. Birnbaum, E., & Lozinskii, E.L. (2003). Consistent subsets of inconsistent systems: structure and behaviour. *Journal of Experimental & Theoretical Artificial Intelligence*, *15*(1), 25–46.
18. Chen, H., Janota, M., & Marques-Silva, J. (2012). QBF-based Boolean function bi-decomposition. In *DATE* (pp. 816–819).
19. Chen, H., & Pál, M. (2004). Optimization, games, and quantified constraint satisfaction. In *Mathematical foundations of computer science* (pp. 239–250).

20. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., & Veith, H. (2003). Counterexample-guided abstraction refinement for symbolic model checking. *Journal of ACM*, *50*(5), 752–794.
21. Condon, A., Feigenbaum, J., Lund, C., & Shor, P.W. (1995). Probabilistically checkable debate systems and nonapproximability of PSPACE-hard functions. *Chicago Journal of Theoretical Computer Science, 1995*.
22. Eén, N., & Sörensson, N. (2003). An extensible SAT-solver. In *SAT* (pp. 502–518).
23. Egly, U., Lonsing, F., & Widl, M. (2013). Long-distance resolution: Proof generation and strategy extraction in search-based QBF solving. In K. L. McMillan, A. Middeldorp, A. Voronkov (Eds.), *LPAR* (Vol. 8312, pp. 291–308). Springer.
24. Fu, Z., & Malik, S. (2006). On solving the partial MAX-SAT problem. In A. Biere, C. P. Gomes (Eds.), *SAT, lecture notes in computer science* (Vol. 4121, pp. 252–265). Springer.
25. Goldberg, E.I., & Novikov, Y. (2003). Verification of proofs of unsatisfiability for CNF formulas. In *DATE* (pp. 10,886–10,891). IEEE Computer Society.
26. Goultiaeva, A., & Bacchus, F. (2010). *Exploiting QBF duality on a circuit representation*. AAAI.
27. Goultiaeva, A., & Bacchus, F. (2013). Recovering and utilizing partial duality in QBF. In *SAT* (pp. 83–99). doi:10.1007/978-3-642-39071-5_8.
28. Goultiaeva, A., Seidl, M., & Biere, A. (2013). Bridging the gap between dual propagation and CNF-based QBF solving. In *DATE* (pp. 811–814).
29. Gupta, A. (2006). Learning abstractions for model checking, Ph.D. thesis, Carnegie Mellon University.
30. Han, B., & Lee, S.J. (1999). Deriving minimal conflict sets by CS-trees with mark set in diagnosis from first principles. *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, *29*(2), 281–286.
31. Heras, F., Morgado, A., & Marques-Silva, J. (2011). Core-guided binary search algorithms for maximum satisfiability. In W. Burgard, D. Roth (Eds.), *AAAI*. AAAI Press.
32. Heule, M., Seidl, M., & Biere, A. (2014). A unified proof system for QBF preprocessing. In S. Demri, D. Kapur, C. Weidenbach (Eds.), *IJCAR* (Vol. 8562, pp. 91–106).
33. Ignatiev, A., Janota, M., & Marques-Silva, J. (2013). Quantified maximum satisfiability: A core-guided approach. In M. Järvisalo, A. Van Gelder (Eds.), *SAT* (Vol. 7962, pp. 250–266). Springer.
34. Ignatiev, A., Morgado, A., Manquinho, V.M., Lynce, I., & Marques-Silva, J. (2014). Progression in maximum satisfiability. In *ECAI* (pp. 453–458). doi:10.3233/978-1-61499-419-0-453.
35. Jain, H., Kroening, D., Sharygina, N., & Clarke, E.M. (2005). Word level predicate abstraction and refinement for verifying RTL verilog. In *DAC* (pp. 445–450). doi:10.1145/1065579.1065697.
36. Janota, M., Grigore, R., & Marques-Silva, J. (2013). On QBF proofs and preprocessing. In K.L. McMillan, A. Middeldorp, A. Voronkov (Eds.), *LPAR* (Vol. 8312, pp. 473–489). Springer.
37. Janota, M., Klieber, W., Marques-Silva, J., & Clarke, E.M. (2012). Solving QBF with counterexample guided refinement. In A. Cimatti, R. Sebastiani (Eds.), *SAT* (Vol. 7317, pp. 114–128). Springer.
38. Janota, M., & Marques-Silva, J. (2011). Abstraction-based algorithm for 2QBF. In *SAT* (pp. 230–244).
39. Kleine Büning, H., & Bubeck, U. (2009). Theory of quantified Boolean formulas. In A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), *Handbook of satisfiability, frontiers in artificial intelligence and applications* (Vol. 185, pp. 735–760). IOS Press.
40. Kleine Büning, H., Karpinski, M., & Flögel, A. (1995). Resolution for quantified Boolean formulas. *Information and Computation*, *117*(1), 12–18.
41. Kleine Büning, H., Subramani, K., & Zhao, X. (2007). Boolean functions as models for quantified Boolean formulas. *Journal of Automated Reasoning*, *39*(1), 49–75.
42. Klieber, W., Sapra, S., Gao, S., & Clarke, E.M. (2010). A non-prenex, non-clausal QBF solver with game-state learning. In *SAT* (pp. 128–142). doi:10.1007/978-3-642-14186-7_12.
43. Kullmann, O. (2011). Constraint satisfaction problems in clausal form II: minimal unsatisfiability and conflict structure. *Fundamental and Information*, *109*(1), 83–119.
44. Liberatore, P. (2005). Redundancy in logic I: CNF propositional formulae. *Artificial Intelligence*, *163*(2), 203–232.
45. Liffiton, M.H., Mneimneh, M.N., Lynce, I., Andraus, Z.S., Marques-Silva, J., & Sakallah, K.A. (2009). A branch and bound algorithm for extracting smallest minimal unsatisfiable subformulas. *Constraints*, *14*(4), 415–442.
46. Liffiton, M.H., & Sakallah, K.A. (2005). On finding all minimally unsatisfiable subformulas. In *SAT* (pp. 173–186).
47. Liffiton, M.H., & Sakallah, K.A. (2008). Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. *Journal of Automated Reasoning*, *40*(1), 1–33.
48. Lonsing, F., & Egly, U. (2014). Incremental QBF solving. In *CP* (pp. 514–530).
49. Lynce, I., & Marques-Silva, J.P. (2004). On computing minimum unsatisfiable cores. In *SAT*.
50. Manquinho, V.M., Marques-Silva, J., & Planes, J. (2009). Algorithms for weighted Boolean optimization. In *SAT* (pp. 495–508).

51. Marques-Silva, J. (2010). Minimal unsatisfiability: models, algorithms and applications (invited paper). In *ISMVL* (pp. 9–14). IEEE Computer Society.
52. Marques-Silva, J., Heras, F., Janota, M., Previti, A., & Belov, A. (2013). On computing minimal correction subsets. In *IJCAI*.
53. Mneimneh, M.N., Lynce, I., Andraus, Z.S., Marques-Silva, J.P., & Sakallah, K.A. (2005). A branch-and-bound algorithm for extracting smallest minimal unsatisfiable formulas. In *SAT* (pp. 467–474).
54. Morgado, A., Dodaro, C., & Marques-Silva, J. (2014). Core-guided MaxSAT with soft cardinality constraints. In *CP* (pp. 564–573). doi:10.1007/978-3-319-10428-7_41.
55. Morgado, A., Heras, F., Liffiton, M.H., Planes, J., & Marques-Silva, J. (2013). Iterative and core-guided MaxSAT solving: a survey and assessment. *Constraints*, *18*(4), 478–534.
56. Narodytska, N., & Bacchus, F. (2014). Maximum satisfiability using core-guided MaxSAT resolution. In *AAAI* (pp. 2717–2723).
57. Niemetz, A., Preiner, M., Lonsing, F., Seidl, M., & Biere, A. (2012). Resolution-based certificate extraction for QBF - (tool presentation). In A. Cimatti, R. Sebastiani (Eds.), *SAT* (Vol. 7317, pp. 430–435). Springer.
58. Nöhrer, A., Biere, A., & Egyed, A. (2012). Managing SAT inconsistencies with HUMUS. In *VAMOS* (pp. 83–91).
59. Plaisted, D.A., & Greenbaum, S. (1986). A structure-preserving clause form translation. *Journal of Symbolic Computation*, *2*(3), 293–304. doi:10.1016/S0747-7171(86)80028-1.
60. Reimer, S., Sauer, M., Marin, P., & Becker, B. (2014). QBF with soft variables. *ECEASST 70*.
61. Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial Intelligence*, *32*(1), 57–95.
62. Roussel, O., & Manquinho, V. (2009). Pseudo-Boolean and cardinality constraints. In A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), *Handbook of satisfiability, frontiers in artificial intelligence and applications* (Vol. 185, pp. 695–733). IOS Press.
63. Ryvchin, V., & Strichman, O. (2011). Faster extraction of high-level minimal unsatisfiable cores. In *SAT* (pp. 174–187).
64. Schaefer, M., & Umans, C. (2002). Completeness in the polynomial-time hierarchy: a compendium. *SIGACT News*, *33*(3), 32–49.
65. Sinz, C., Kaiser, A., & Küchlin, W. (2003). Formal methods for the validation of automotive product configuration data. *AI EDAM*, *17*(1), 75–97.
66. Stuckey, P.J. (2013). There are no CNF problems. In *SAT* (pp. 19–21). doi:10.1007/978-3-642-39071-5_3.
67. Stuckey, P.J., Sulzmann, M., & Wazny, J. (2003). Interactive type debugging in Haskell. In *ACM SIGPLAN workshop on Haskell* (pp. 72–83). ACM.
68. Tseitin, G.S. (1968). On the complexity of derivations in the propositional calculus. *Studies in Mathematics and Mathematical Logic Part II*, 115–125.
69. Van Gelder, A. (2012). Contributions to the theory of practical quantified Boolean formula solving. In M. Milano (Ed.), *CP* (Vol. 7514, pp. 647–663). Springer.
70. Yu, Y., & Malik, S. (2005). Validating the result of a quantified Boolean formula (QBF) solver: theory and practice. In *ASP-DAC* (pp. 1047–1051).
71. Zhang, J., Li, S., & Shen, S. (2006). Extracting minimum unsatisfiable cores with a greedy genetic algorithm. In *AUS-AI* (pp. 847–856).
72. Zhang, L. (2006). Solving QBF by combining conjunctive and disjunctive normal forms. In *AAAI* (pp. 143–150).
73. Zhang, L., & Malik, S. (2002). Conflict driven learning in a quantified Boolean satisfiability solver. In *ICCAD* (pp. 442–449).
74. Zhang, L., & Malik, S. (2003). Validating SAT solvers using an independent resolution-based checker: practical implementations and other applications. In *DATE* (pp. 10,880–10,885). IEEE Computer Society.