Quantified Maximum Satisfiability:
A Core-Guided Approach

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

[1] IST/INESC-ID, Lisbon, Portugal
[2] University College Dublin, Ireland
[3] ISDCT SB RAS, Irkutsk, Russia
{aign,mikolas}@sat.inesc-id.pt, jpms@ucd.ie

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 the problem of solving quantified formulas subject to a cost function, and considers the problem in a quantified MaxSAT setting. Two approaches are investigated. One is based on relaxing the soft clauses and performing a linear 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 concrete problem of computing smallest minimal unsatisfiable subformulas (SMUS), a decision version of which is a well-known problem in the second level of the polynomial hierarchy. Experimental results, obtained on representative problem instances, indicate that the core-guided approach for the SMUS problem outperforms the use of linear search over the values of the cost function. More significantly, the core-guided approach also outperforms the state-of-the-art SMUS extractor Digger.

1 Introduction

When reasoning about quantified Boolean formulas (QBF), different optimization problems can be envisioned. MAX-QSAT [16] 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 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 in quantified CSP involves computing strategies that optimize some cost function or associating costs with strategies [14,8]. 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 values in [13]). Many other concrete examples are given by the optimization versions of decision problems in the polynomial hierarchy [35].

M. Järvisalo and A. Van Gelder (Eds.): SAT 2013, LNCS 7962, pp. 250–266, 2013.

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 [19,2,23,9], 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 [19,9]. In contrast, binary search refines both lower and upper bounds [19,23]. 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 [19]. 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 [25,24] can be modified to produce unsatisfiable cores.

The new algorithms for QMaxSAT are evaluated on the problem of computing the *smallest minimally unsatisfiable subformula* (SMUS) [29,32]. The SMUS decision problem is well-known to be in the second level of the polynomial hierarchy (e. g. [20]) and studied in the context of formal verification. Computing SMUSes is also relevant for assessing the quality of computed MUSes in practice. The third contribution of the paper is a novel QMaxSAT formulation for the SMUS problem, and QMaxSAT-based algorithm. 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 [26]. More importantly, these results validate the use of core-guided approaches for QMaxSAT.

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 completemented by a description in Section 4.1 of how a CEGAR-based 2QBF instrumented to generate unsatisfiable cores, and so used in QMaxSAT algorithms. Section 5 shows the practicality of the framework: it models the SMUS problem as QMaxSAT and describes improvements to the QMaxSAT for the concrete problem of computing an SMUS. Section 6 presents the experimental results on computing SMUSes. 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 an atomic 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 $\varphi = c_1 \land c_2 \land \ldots \land c_m$, $m \in \mathbb{N}$, 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. The notion of assignment $\mathcal{A}_X$ can be extended to literals by setting $\mathcal{A}_X(\neg x_i) = 1 - \mathcal{A}_X(x_i)$ for $x_i \in X$. Hereinafter, expression $\varphi|_{\mathcal{A}_X}$ denotes a formula obtained from a Boolean formula φ by replacing each variable x_i of X with its value $\mathcal{A}_X(x_i)$. The same *restriction* notation $c|_{\mathcal{A}_X}$ is used with regard to a clause c of a CNF formula. Since formula φ expresses some Boolean function $f(x_1, \ldots, x_n)$, $\varphi|_{\mathcal{A}_X}$ defines the value of function f, which in what follows is denoted by $f(\mathcal{A}_X)$. The same notation is used for denoting values of pseudo-Boolean functions.

If $\mathcal{A}_X(l_i) = 1$ then literal l_i is said to be *satisfied* by assignment $\mathcal{A}_X$; if $\mathcal{A}_X(l_i) = 0$ then l_i is *falsified* by $\mathcal{A}_X$. Assignment $\mathcal{A}_X$ satisfies a clause c, i. e. $c|_{\mathcal{A}_X} = 1$, if it satisfies at least one of its literals; otherwise the clause is said to be falsified by $\mathcal{A}_X$ ($c|_{\mathcal{A}_X} = 0$). If for a given CNF formula φ there is an assignment $\mathcal{A}_X$ such that $\varphi|_{\mathcal{A}_X} = 1$, then formula φ is called *satisfiable* and $\mathcal{A}_X$ is its *satisfying assignment*, or *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 such an assignment $\mathcal{A}_X$ that satisfies the maximum number of clauses of φ. A standard way of dealing with MaxSAT problems is to introduce a set $R = \{r_1, r_2, \ldots, r_m\}$ of additional variables (called *relaxation variables*) and consider a relaxed CNF formula $\varphi_R = \{c_1^R, \ldots, c_m^R\}$, where $c_i^R = c_i \vee r_i$. Observe that φ_R is satisfiable. The MaxSAT problem for φ can be now formulated as follows: given a cost function $f(r_1, \ldots, r_m) = \sum_{i=1}^{m} r_i$, find an assignment $\mathcal{A}_{X \cup R} \in \mathcal{M}(\varphi_R)$ such that for any other assignment $\mathcal{B}_{X \cup R} \in \mathcal{M}(\varphi_R)$

$$f(\mathcal{A}_{X \cup R}) \leq f(\mathcal{B}_{X \cup R})$$

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 $\mathcal{A}_X$ that satisfies all the hard clauses and maximizes the number of the soft clauses that are satisfied. Analogously to the MaxSAT formulation given above, one can formulate the partial MaxSAT problem by relaxing only the soft clauses and considering a cost function using the corresponding relaxation variables.

2.3 Quantified Boolean Formula

Quantified Boolean formulas (QBFs) are an extension of propositional logic with *existential* and *universal* quantifiers ($\forall$, $\exists$) [11].

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 using only 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 if and only if

$Q_2x_2 \ldots Q_nx_n.\ \varphi|_{x_1=1}$ or $Q_2x_2 \ldots Q_nx_n.\ \varphi|_{x_1=0}$ is true. A QBF $\forall x_1 Q_2x_2 \ldots Q_nx_n.\ \varphi$ is true iff $Q_2x_2 \ldots Q_nx_n.\ \varphi|_{x_1=1}$ and $Q_2x_2 \ldots Q_nx_n.\ \varphi|_{x_1=0}$ are true. To decide whether a given QBF is true or not, is PSPACE-complete [11].

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_jX_j instead of $Q_jx_1 \ldots Q_jx_n$. Therefore, a formula with k quantification levels can be denoted by $Q_1X_1 \ldots Q_kX_k.\ \varphi$. Moreover, the prefix $Q_1X_1 \ldots Q_kX_k$ of a QBF with k quantification levels is usually denoted by $\overrightarrow{Q}$.

In Section 5, devoted to the SMUS problem, we focus on QBFs with 2 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 [11].

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 $\mathcal{M}(\psi)$.

3 Quantified MaxSAT

In this section we consider an optimization formulation of the QBF problem, when one should choose such a solution of the problem (among all solutions), that is optimal with respect to some given criterion. This kind of problems is a natural generalization of MaxSAT: instead of CNF formulas, we consider quantified formulas specified in a general form. Moreover, the optimization criterion in this problem is generalized as well. For example, it is possible to specify it as a minimization problem for some pseudo-Boolean cost function (see [21]).

Consider sets of Boolean variables X_1, X_2, …, X_k and a set of additional variables $E = \{e_1, \ldots, e_l\}$. Let

$$\psi = \exists E\ \overrightarrow{Q}.\ \varphi \tag{1}$$

be a quantified Boolean formula, where its matrix φ is a propositional formula over the set $(\bigcup_{i=1}^{k} X_i) \cup E$ given in a potentially non-CNF form. Consider a linear[2] pseudo-Boolean function $f(e_1, \ldots, e_l) = \sum_{i=1}^{l} a_i \cdot e_i$ as a cost function. Then the quantified MaxSAT (QMaxSAT) problem can be formulated as the problem of finding an assignment $\mathcal{A}_E \in \mathcal{M}(\psi)$ such that for any other assignment $\mathcal{B}_E \in \mathcal{M}(\psi)$

$$f(\mathcal{A}_E) \leq f(\mathcal{B}_E)$$

Example 1. Consider a 2QBF formula

$$\xi = \exists e_1, e_2\ \forall x_1, x_2.\ \varphi,$$

[1] Note that solution of a quantified formula defined in this way represents a "portion" of the formula's *model*, which is defined, for example, in [12].

[2] Non-linear pseudo-Boolean formulas can be linearized with the use of auxiliary variables. Some linearization techniques are described in [18,6].

where $\varphi = (\neg e_1 \land \neg e_2) \rightarrow (x_1 \lor x_2)$; and a cost function $f(e_1, e_2) = 2 \cdot e_1 + 3 \cdot e_2$. Formula ξ has three possible solutions: $e_1 = 0, e_2 = 1$; $e_1 = 1, e_2 = 0$; $e_1 = 1, e_2 = 1$. However, the optimal solution which minimizes the cost function is $e_1 = 1, e_2 = 0$, i. e. $f(1, 0) = \min_{\mathcal{M}(\xi)} f(e_1, e_2) = 2$.

Let us show how the formulated QMaxSAT problem relates to classical (quantifier-free) MaxSAT. First, we define another problem, which is a special case of QMaxSAT. Consider a QBF $\overrightarrow{Q}.\ \varphi$, which is false, and let its matrix φ be in CNF. Then the problem of finding a maximal subset $\varphi' \subset \varphi$ such that $\overrightarrow{Q}.\ \varphi'$ is true, can be easily expressed in terms of QMaxSAT described above. To do this, one should consider a set R of relaxation variables and a CNF formula $\varphi_R = \{c_1 \lor r_1, \ldots, c_m \lor r_m\}$, $c_i \in \varphi$, $r_i \in R$, and choose $f(r_1, \ldots, r_m) = \sum_{i=1}^{m} r_i$ as the cost function. Then the problem is to find the *best* solution of QBF $\exists R\, \overrightarrow{Q}.\ \varphi_R$ subject to the cost function f. This problem is obviously a generalization of classical MaxSAT but also a special case of QMaxSAT. Note that although variables E from the QMaxSAT formulation are replaced by relaxation variables R here, they do not play the role of relaxation variables in general (e. g., see the matrix of formula ξ in Example 1).

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 an explanation of how MaxSAT algorithms can be adapted to the QMaxSAT problem.

Related Work. Optimization problems subject to quantified constraints have been studied elsewhere [16,14,8], but address more general formulations than QMaxSAT. The Max-QSAT problem [16] 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 [14,8]. The focus of [14] is approximation algorithms for computing a winning strategy that minimizes some cost function, whereas [8] studies preferences over strategies. To our best knowledge, and besides our work, [8] is the only other reference that proposes an exact algorithm for solving optimization problems over quantified constraints.

4 QMaxSAT Algorithms

One of the simplest approaches to the QMaxSAT problem is to iteratively decide the following formula with a QBF oracle:

$$\exists E\, \overrightarrow{Q}.\ \varphi \land (f(e_1, \ldots, e_l) \leq k) \tag{2}$$

Here one can start from a lower bound (e. g. $k = 0$) and increase k until formula (2) becomes true, or decrease it from some upper bound (e. g. $k = \max_{\{0,1\}^l} f$) value while (2) is true. This is analogous to the linear search for non-quantified MaxSAT [9], which refines lower and upper bounds on the value of the cost function[3]. Although these

[3] Instead of the linear search algorithms, one can use binary search [19,23]. Binary search algorithms are not covered by this paper.

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).

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 [19]. 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 [30]; some authors refer to this algorithm as WPM1 [1]) tests a series of unsatisfiable instances until a satisfiable instance is found. However, instead of dealing with the constraint $f(e_1, \ldots, e_l) \leq k$ and increasing k with each call to a SAT solver, it identifies a small unsatisfiable part of the current formula, 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.

Recall that function f is linear, i. e. $f(e_1, \ldots, e_l) = \sum_{i=1}^{l} a_i \cdot e_i$. Assume[4], that $a_i = 1, \forall i \in \{1, \ldots, l\}$. For each term e_i of formula $f(e_1, \ldots, e_l) = \sum_{i=1}^{l} e_i$ create a unit clause $\neg e_i$. Denote CNF formula $\{\neg e_1, \neg e_2, \ldots, \neg e_l\}$ by φ_S. Observe that each term e_i of f incrementing its value (i. e. $e_i = 1$) corresponds to a falsified clause $\neg e_i$ of φ_S. This means that an assignment evaluates function f to some value y, $0 \leq y \leq l$, if and only if it satisfies $l - y$ clauses of φ_S. Therefore, function f is evaluated to its minimum value by such an assignment $\mathcal{A}_E$, that maximizes the number of satisfied clauses of φ_S. Let $\#(\varphi_S|_{\mathcal{A}_E})$ be a function that outputs the number of clauses of φ_S that are satisfied by some assignment $\mathcal{A}_E$, i. e. $\#(\varphi_S|_{\mathcal{A}_E}) = \sum_{c \in \varphi_S} c|_{\mathcal{A}_E}$. Instead of QBF ψ (see (1)), consider the formula

$$\psi' = \exists E \overrightarrow{Q}. \varphi \wedge \varphi_S \tag{3}$$

Now we can formulate another way to solve the QMaxSAT problem for QBF ψ subject to the cost function f. It consists in finding an assignment $\mathcal{A}_E \in \mathcal{M}(\psi)$ such that for any other assignment $\mathcal{B}_E \in \mathcal{M}(\psi)$ the following holds: $\#(\varphi_S|_{\mathcal{A}_E}) \geq \#(\varphi_S|_{\mathcal{B}_E})$. On the analogy of partial MaxSAT, CNF φ_S can be treated as a set of *soft* clauses while the original QBF matrix φ is a *hard* part given in a potentially non-CNF form. Let us define a core of formula ψ. This will enable us to extend the MSU1 algorithm to the case of QMaxSAT.

Definition 1. *A Boolean formula $\varphi_C = \varphi \wedge \varphi'_S$, $\varphi'_S \subseteq \varphi_S$, is called an unsatisfiable core of formula ψ', if and only if the following is false*

$$\exists E \overrightarrow{Q}. \varphi_C$$

According to Definition 1, the hard part of formula ψ' is included into any unsatisfiable core φ_C of ψ'. However, similarly to the core-guided algorithms for the non-quantified MaxSAT, in the algorithm described below we will need only the soft part φ'_S of the core. The algorithm selects soft clauses from the core by calling a function $\texttt{Soft}(\varphi_C)$.

Algorithm 1 shows the pseudo-code of the MSU1 algorithm adapted to QMaxSAT (we refer to this algorithm as *QMSU1*). For a formula ψ' given in the form (3), which is

[4] Otherwise we have a *weighted* version of the problem, and all the ideas described in this section, can be extended as is done for weighted MaxSAT algorithms [30,1].

Algorithm 1. QMSU1 Algorithm

input : A QBF $\psi = \exists E \overrightarrow{Q}.\, \varphi$ s. t. $\mathcal{M}(\psi) \neq \emptyset$, and a CNF φ_S
output : $\mathcal{A}_E \in \mathcal{M}(\psi)$, s. t. $\forall \mathcal{B}_E \in \mathcal{M}(\psi)$: $\#(\varphi_S|_{\mathcal{A}_E}) \geq \#(\varphi_S|_{\mathcal{B}_E})$

1 $R_{all} \leftarrow \emptyset$ `// set of all relaxation variables`
2 **while** true **do**
3 $\quad \psi'_R = \exists E\, \exists R_{all}\, \overrightarrow{Q}.\, \varphi \wedge \varphi_S$
4 $\quad (\mathsf{st}, \varphi_C, \mathcal{A}_E) \leftarrow \mathtt{QBF}(\psi'_R)$ `// calling a QBF oracle`
5 $\quad$ **if** st $=$ true **then**
6 $\quad\quad$ **return** $\mathcal{A}_E$
7 $\quad R \leftarrow \emptyset$ `// set of relaxation variables`
8 $\quad$ **foreach** $c \in \mathtt{Soft}(\varphi_C)$ **do**
9 $\quad\quad$ **let** r be a new relaxation variable
10 $\quad\quad R \leftarrow R \cup \{r\}$
11 $\quad\quad \varphi_S \leftarrow \varphi_S \setminus \{c\} \cup \{c \vee r\}$
12 $\quad \varphi \leftarrow \varphi \wedge \mathtt{CNF}(\sum_{r \in R} r \leq 1)$
13 $\quad R_{all} \leftarrow R_{all} \cup R$

implicitly defined by a QBF ψ from (1) and a CNF formula φ_S, the QMSU1 algorithm outputs such a solution $\mathcal{A}_E$ of ψ that maximizes the number of satisfied clauses of φ_S over the set $\mathcal{M}(\psi)$. One important pre-condition of the algorithm is that formula ψ must have at least one solution, i. e. $\mathcal{M}(\psi) \neq \emptyset$. 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 relaxed copy ψ'_R of formula ψ' (line 3) and asks a QBF oracle to decide whether it is true or false (line 4). As an answer the oracle returns a 3-tuple $(\mathsf{st}, \varphi_C, \mathcal{A}_E)$. If $\mathsf{st} = \mathsf{false}$, then the algorithm considers a set of relaxation variables R (initially set to $\emptyset$) and processes the unsatisfiable core φ_C returned by the QBF oracle. This step consists of relaxing soft clauses of the core, i. e. the algorithm introduces a new relaxation variable $r \in R$ for each soft clause c of the core φ_C, and replaces original clause c with its relaxed copy $c \vee r$ 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 of ψ'. Note that since each relaxation variable $r \in R_{all}$ is added only to a clause of the form $\neg e_j$, all of them can be quantified by the same $\exists$-quantifier as variables $e_j \in E$ (see line 3). The algorithm iterates until formula ψ'_R is true and $\mathcal{A}_E \in \mathcal{M}(\psi'_R)$ (line 6). By construction, $\mathcal{A}_E$ maximizes the number of satisfied clauses of φ_S over the set $\mathcal{M}(\psi)$, i. e. it is the solution of a 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 [19].

Note that the only requirement imposed by the QMSU1 algorithm on the QBF oracle is the ability to produce a certificate that could validate the answer *true* or *false*. 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

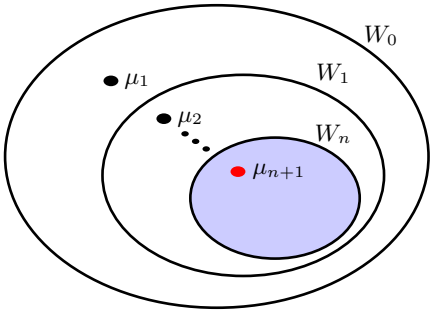


Fig. 1. Gradual strengthening of abstractions until a solution is found

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. There exist methods for extracting unsatisfiable cores from unsatisfiable QBF instances for DPLL-based QBF solvers (e. g., see [37]). While the QMSU1 algorithm can use any QBF solver as long as it produces cores, this work uses a CEGAR-based 2QBF solver [25,24] as an underlying QBF oracle for a particular problem (the smallest MUS problem). Section 4.1 describes a method using a CEGAR-based 2QBF solver for extracting unsatisfiable cores. Following the ideas of [24], the method can be easily extended to the case of formulas with an arbitrary number of quantification levels. The task is further simplified by the fact that the QMSU1 algorithm requires only soft part of the core, which depends on variables quantified at the outermost level.

4.1 Extracting Cores in CEGAR-Based 2QBF

Among the many practical uses of the *counterexample guided abstraction refinement* (CEGAR) [15], it can also be applied for solving 2QBF [25]. 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[5] and describes its modification, which is able to extract an unsatisfiable core of a formula if the formula is false.

For the sake of succintness, in this section we denote assignments to variables of X and Y by μ and ν, respectively. We also assume, that the matrix of the 2QBF is presented as $\varphi_H \wedge \varphi_S$, 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 $\exists X \forall Y.\, \varphi_H \wedge \varphi_S$ can be equivalently represented as

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

where the universal quantifier is *expanded* using a conjunction. Since the full expansion (4) of the problem can be exponentially large with respect to the original problem,

[5] The reader is referred to [25] for further details and properties of the algorithm.

Algorithm 2. CEGAR loop for 2QBF

input : $\exists X \forall Y.\ \varphi_H \wedge \varphi_S$
output : (true, μ) if there exists μ s.t. $\forall Y.\ (\varphi_H \wedge \varphi_S)|_\mu$,
$(\mathsf{false}, \varphi_H \wedge \varphi'_S)$ s.t. $\varphi'_S \subseteq \varphi_S$ otherwise

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

it is infeasible to construct such representation in practice. Instead of constructing the full expansion (4), CEGAR constructs a *partial expansion* of the given problem, i. e.

$$\exists X.\ \bigwedge_{\nu \in W} (\varphi_H \wedge \varphi_S)|_\nu \tag{5}$$

where $W \subseteq \{0,1\}^{|Y|}$. We refer to formula (5) as W*-abstraction*. Observe that for any W, the corresponding W-abstraction is weaker than the full expansion (4). 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 (4). 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 Figure 1).

Algorithm 2 shows the pseudocode of the algorithm. The algorithm maintains a set of assignments W in the variable ω. We start with the abstraction equal to the formula 1, any assignment μ to the variable of X satisfies the abstraction. Assume, that the algorithm encodes the hard part φ_H into a CNF formula by calling a function $\mathtt{CNF}(\varphi_H)$.

In each iteration of the loop, the algorithm first computes a solution to the abstraction, which is maintained in φ 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 in the form $\varphi_H \wedge \varphi'_S$ (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: φ'_S should include a clause $c \in \varphi_S$ if there is a clause $c' \in \varphi_C$ and an assignment $\nu \in \omega$ such that $c' = c|_\nu$. 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. $\exists X.\ \bigwedge_{\nu \in W}(\varphi_H \wedge \varphi'_S)|_\nu$ is false. Note that there can be several clauses $c_i \in \varphi_S$ such that $c' = c_i|_\nu$. However, it is sufficient to include just one of these clauses into φ'_S — by doing this one can get smaller QBF cores.

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 $\neg\varphi|_{\mu,\nu}$. A counterexample ν serves as a witness that μ is not a solution, i. e. it is not the case that $\forall Y.\ \varphi|_\mu$ because φ is false when y has the value 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.

5 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 $\psi \subseteq \varphi$ is called a *minimal unsatisfiable subformula* (MUS) of φ, if ψ is unsatisfiable and $\forall c_i \in \psi$ formula $\psi \setminus \{c_i\}$ is satisfiable. The MUS problem is a subject of active research (e. g. [31]).

Definition 2. *Formula ψ^*, $\psi^* \subseteq \varphi$, is called a smallest MUS of φ if*

1. *ψ^* is unsatisfiable;*
2. *for any MUS ψ, $\psi \subseteq \varphi$, the following holds $|\psi^*| \leq |\psi|$.*

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 [29]. A greedy genetic algorithm that finds approximate solutions of the SMUS problem was proposed in [38]. A branch and bound algorithm for computing SMUSes was described in [32,26]. 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 [20]). The Digger algorithm, which is a state-of-the-art algorithm for computing an SMUS of a CNF formula, was proposed in [32,26].

Let us formulate an optimization extension of SMUS in terms of QMaxSAT defined in Section 3. First, we consider a set of *selection variables* $S = \{s_1, \ldots, s_m\}$. Instead of formula φ, we consider a formula $\varphi_R = \{c_1 \vee \neg s_1, \ldots, c_m \vee \neg s_m\}$, $c_i \in \varphi$. Let us introduce a function $f : \{0,1\}^m \to \mathbb{N}$:

$$f(s_1, \ldots, s_m) = \sum_{i=1}^{m} s_i.$$

Then the problem of finding a smallest MUS of φ consists in finding such an assignment $\mathcal{A}_S \in \mathcal{M}(\neg\varphi_R)$ that for any other assignment $\mathcal{B}_S \in \mathcal{M}(\neg\varphi_R)$ the following holds: $f(\mathcal{A}_S) \leq f(\mathcal{B}_S)$.

As shown in Section 4, 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:

$$\exists S \forall X.\ \neg\varphi_R \wedge (f(s_1, \ldots, s_m) \leq k) \tag{6}$$

However, one can apply algorithm QMSU1 to this problem as well. Similarly to Section 4, we introduce a set $\varphi_s = \{\neg s_1, \neg s_2, \ldots, \neg s_m\}$ of soft clauses instead of considering constraint $f(s_1, \ldots, s_m) \leq k$ and iteratively ask the QBF oracle to decide the following QBF:

$$\psi = \exists S \forall X.\ \neg\varphi_R \wedge \varphi_S \tag{7}$$

Observe that CNF formula φ_S is the set of soft clauses of ψ while $\neg\varphi_R$ is the hard part presented in a non-clausal form. Thus, the QMSU1 algorithm iteratively extracts unsatisfiable cores of formula ψ and relaxes their soft parts, which are some subsets of φ_S, until it finds an assignment $\mathcal{A}_S \in \mathcal{M}(\neg\varphi_R)$ that maximizes the number of satisfied clauses of φ_S. Assignment $\mathcal{A}_S$ corresponds to an SMUS ψ^*, $\psi^* \subseteq \varphi$, such that a clause $c_i \in \psi^*$ iff $\mathcal{A}_S(s_i) = 1$.

5.1 Improvements to the Algorithm

To increase the performance of the Digger algorithm, the authors of [26] 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 $c \in \delta$. There is an important connection between MCSes and MUSes of CNF formulas (see [34,22,10,3,27,28]): any MUS of formula φ is a *minimal 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.

Due to the fact that the QMSU1 algorithm does not handle constraints $\leq k$ directly, lower bounds for SMUS themselves cannot be directly used in QMSU1. 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. Therefore, one of the improvements of QMSU1 for computing an SMUS of formula φ can be enumeration of all the unit MCSes of φ.

Another technique we exploit in our approach is the use of 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 formula

$$\exists S \forall X.\ \neg\varphi_R \wedge \varphi_S^\delta$$

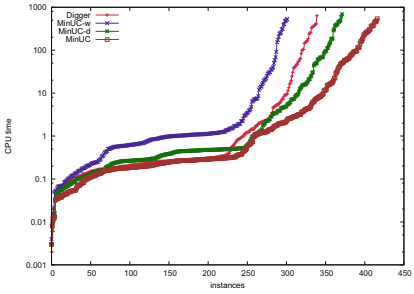
is false. Given Definition 1, this means that $\neg\varphi_R \wedge \varphi_S^\delta$ is a core of (7). Therefore, k MCSes computed by preprocessing give us k unsatisfiable cores of (7). Moreover, since all the computed MCSes are disjoint, the cores are disjoint. In practice, the use of this preprocessing technique significantly increases the performance of the QMSU1 algorithm.

6 Experimental Results

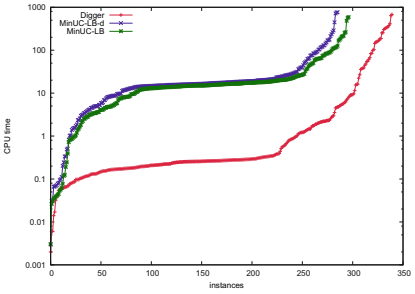
A prototype of a solver for the SMUS problem implementing the QMSU1 algorithm was developed with the use of a CEGAR-based 2QBF oracle described in Section 4.1. The underlying SAT solver of our 2QBF oracle implementation is MINISAT 2.2 [17]. We refer to this prototype as *MinUC* (**Min**imum **U**nsatisfiable **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.

During the course of this research, we implemented a number of efficient algorithms to enumerate disjoint MCSes of CNF formulas. These are beyond the scope of this paper. However, to do a more comprehensive comparison between QMSU1 and Digger, we ran MinUC in three different modes (the corresponding names of the tools are presented in the parentheses):

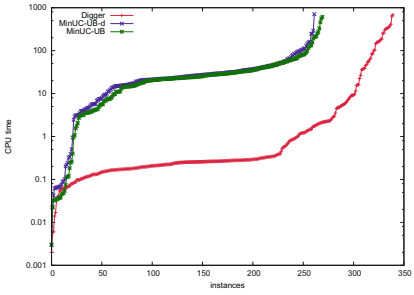
- without enumerating disjoint MCSes (*MinUC-w*);
- with the use of the Digger's disjoint MCS enumerator (*MinUC-d*);
- with the use of the default built-in disjoint MCS enumerator (*MinUC*).



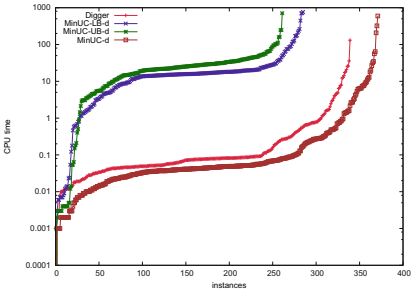
(a) QMSU1 mode



(b) Linear LB mode

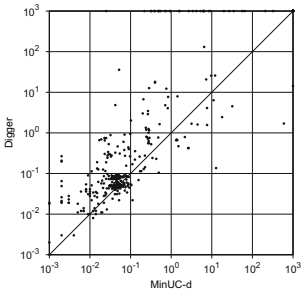


(c) Linear UB mode

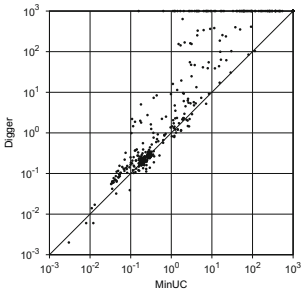


(d) All modes with Digger's MCS enumerator

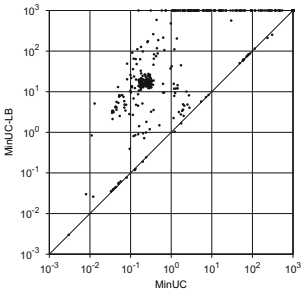
Fig. 2. Performance of MinUC compared to Digger



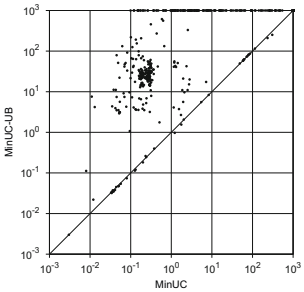
(a) MinUC-d vs Digger



(b) MinUC vs Digger



(c) MinUC vs MinUC-LB



(d) MinUC vs MinUC-UB

Fig. 3. Performance of the used approaches

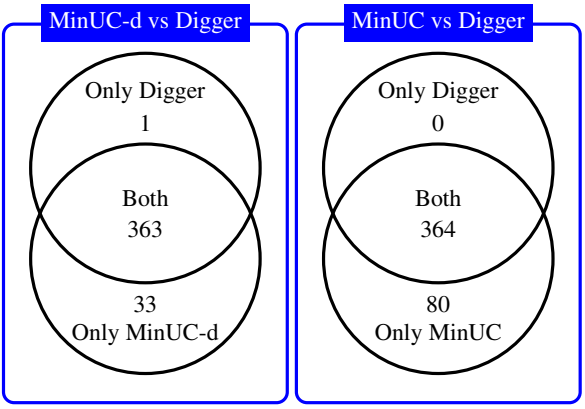


Fig. 4. Number of instances solvable by different solvers: Digger vs MinUC

The set of instances considered includes several sets of benchmarks described below. The first set consists of automotive product configuration benchmarks [36]. Two other sets of benchmarks come from circuit diagnosis. Additionally, we selected instances from the complete set of the MUS competitions benchmarks[6] as follows. Since the SMUS problem is computationally harder than problem of extracting an MUS of formula, we picked all the instances from the MUS competitions that can be solved by muser-2 (see [7]) in 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 seconds 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 Figure 2c 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 Figure 3a and Figure 3b) and to its linear search versions (Figure 3c and Figure 3d). Figure 4 gives an overview on how many instances are solvable either by Digger or by core-guided MinUC only.

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 of possible improvements.

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) problem; and (iv) new pruning techniques for solving the SMUS problem. The novel core-guided algorithm for QMaxSAT is based on

[6] `http://www.satcompetition.org/2011/`

the original work of Fu&Malik [19]. Nevertheless, other algorithms for non-quantified MaxSAT can also be adapted to the quantified case (e. g. [2,23]).

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 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 [16] among others [14,8]. 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 [37] and then followed by recent works on certificate generation for resolution-based QBF solvers (e. g. [4,5,33]). 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.

For the concrete application of QMaxSAT, the SMUS problem, several optimizations can be considered. Modern (and efficient) MUS solvers [7] can be used for computing an upper bound on the size of the SMUS. If the lower bound (e. g. due to disjoint cores, or by iterative core extraction) equals the upper bound, then an SMUS 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 multiannual PIDDAC program funds (PEst-OE/EEI/LA-0021/2011).

References

1. Ansótegui, C., Bonet, M.L., Levy, J.: Solving (Weighted) Partial MaxSAT through Satisfiability Testing. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 427–440. Springer, Heidelberg (2009)
2. Ansótegui, C., Bonet, M.L., Levy, J.: A new algorithm for weighted partial MaxSAT. In: AAAI Conference on Artificial Intelligence. AAAI (2010)
3. Bailey, J., Stuckey, P.J.: Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. In: Hermenegildo, M.V., Cabeza, D. (eds.) PADL 2004. LNCS, vol. 3350, pp. 174–186. Springer, Heidelberg (2005)
4. Balabanov, V., Jiang, J.H.R.: Resolution proofs and skolem functions in QBF evaluation and applications. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 149–164. Springer, Heidelberg (2011)

5. Balabanov, V., Jiang, J.H.R.: Unified qbf certification and its applications. Formal Methods in System Design 41(1), 45–65 (2012)
6. Balas, E., Mazzola, J.: Nonlinear 0–1 programming: I. Linearization techniques. Mathematical Programming 30, 1–21 (1984), http://dx.doi.org/10.1007/BF02591796, 10.1007, doi:10.1007/BF02591796
7. Belov, A., Lynce, I., Marques-Silva, J.: Towards efficient MUS extraction. AI Commun. 25(2), 97–116 (2012)
8. Benedetti, M., Lallouet, A., Vautard, J.: Quantified constraint optimization. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 463–477. Springer, Heidelberg (2008)
9. Berre, D.L., Parrain, A.: The Sat4j library, release 2.2. JSAT 7(2-3), 59–64 (2010)
10. Birnbaum, E., Lozinskii, E.L.: Consistent subsets of inconsistent systems: structure and behaviour. J. Exp. Theor. Artif. Intell. 15(1), 25–46 (2003)
11. Büning, H.K., Bubeck, U.: Theory of quantified boolean formulas. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, pp. 735–760. IOS Press (2009)
12. Büning, H.K., Subramani, K., Zhao, X.: Boolean Functions as Models for Quantified Boolean Formulas. J. Autom. Reasoning 39(1), 49–75 (2007)
13. Chen, H., Janota, M., Marques-Silva, J.: QBF-based Boolean function bi-decomposition. In: Design, Automation & Test in Europe Conference, pp. 816–819 (2012)
14. Chen, H., Pál, M.: Optimization, games, and quantified constraint satisfaction. In: Fiala, J., Koubek, V., Kratochvíl, J. (eds.) MFCS 2004. LNCS, vol. 3153, pp. 239–250. Springer, Heidelberg (2004)
15. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
16. Condon, A., Feigenbaum, J., Lund, C., Shor, P.W.: Probabilistically checkable debate systems and nonapproximability of PSPACE-hard functions. Chicago J. Theor. Comput. Sci. (1995)
17. Eén, N., Sörensson, N.: An Extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT 2003. LNCS, vol. 2919, pp. 502–518. Springer, Heidelberg (2004)
18. Fortet, R.: Applications de l'algèbre de Boole en recherche opérationnelle. Revue Française d'Informatique et de Recherche Opérationnelle 4, 17–26 (1960)
19. Fu, Z., Malik, S.: On solving the partial MAX-SAT problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 252–265. Springer, Heidelberg (2006)
20. Gupta, A.: Learning Abstractions for Model Checking. Ph.D. thesis, Carnegie Mellon University (June 2006)
21. Hammer, P., Rudeanu, S.: Boolean Methods in Operations Research and Related Areas. Springer (1968)
22. Han, B., Lee, S.J.: 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 (1999)
23. Heras, F., Morgado, A., Marques-Silva, J.: Core-guided binary search for maximum satisfiability. In: AAAI Conference on Artificial Intelligence. AAAI (2011)
24. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.: Solving QBF with counterexample guided refinement. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 114–128. Springer, Heidelberg (2012)
25. Janota, M., Marques-Silva, J.: Abstraction-based algorithm for 2QBF. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 230–244. Springer, Heidelberg (2011)
26. Liffiton, M.H., Mneimneh, M.N., Lynce, I., Andraus, Z.S., Marques-Silva, J., Sakallah, K.A.: A branch and bound algorithm for extracting smallest minimal unsatisfiable subformulas. Constraints 14(4), 415–442 (2009)

27. Liffiton, M.H., Sakallah, K.A.: On Finding All Minimally Unsatisfiable Subformulas. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 173–186. Springer, Heidelberg (2005)
28. Liffiton, M.H., Sakallah, K.A.: Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. J. Autom. Reasoning 40(1), 1–33 (2008)
29. Lynce, I., Marques-Silva, J.P.: On Computing Minimum Unsatisfiable Cores. In: SAT (2004)
30. Manquinho, V., Marques-Silva, J., Planes, J.: Algorithms for weighted boolean optimization. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 495–508. Springer, Heidelberg (2009)
31. Marques-Silva, J.: Minimal unsatisfiability: Models, algorithms and applications (invited paper). In: ISMVL, pp. 9–14. IEEE Computer Society (2010)
32. Mneimneh, M., Lynce, I., Andraus, Z., Marques-Silva, J., Sakallah, K.: A Branch-and-Bound Algorithm for Extracting Smallest Minimal Unsatisfiable Formulas. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 467–474. Springer, Heidelberg (2005)
33. Niemetz, A., Preiner, M., Lonsing, F., Seidl, M., Biere, A.: Resolution-Based Certificate Extraction for QBF - (Tool Presentation). In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 430–435. Springer, Heidelberg (2012)
34. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. 32(1), 57–95 (1987)
35. Schaefer, M., Umans, C.: Completeness in the polynomial-time hierarchy: a compendium. SIGACT News 33(3), 32–49 (2002)
36. Sinz, C., Kaiser, A., Küchlin, W.: Formal methods for the validation of automotive product configuration data. AI EDAM 17(1), 75–97 (2003)
37. Yu, Y., Malik, S.: Validating the result of a Quantified Boolean Formula (QBF) solver: theory and practice. In: Asia and South Pacific Design Automation Conference, pp. 1047–1051 (2005)
38. Zhang, J., Li, S., Shen, S.: Extracting minimum unsatisfiable cores with a greedy genetic algorithm. In: Australian Conference on Artificial Intelligence, pp. 847–856 (2006)