Quantified Maximum Satisfiability: A Core-Guided Approach

Alexey Ignatiev¹, Mikoláš Janota¹, and Joao Marques-Silva¹,²

¹ INESC-ID/IST, Lisbon, Portugal
² CASL/CSI, University College Dublin, Ireland

Sixteenth International Conference on Theory and Applications of Satisfiability Testing
Helsinki, Finland
July 11, 2013
Motivation

Many practical decision problems can be represented as QBF.
Motivation

Many practical *decision* problems can be represented as QBF.

**Smallest MUS problem**

Find a *smallest* unsatisfiable subformula of a CNF formula.

Decision version (for $\varphi$ and $k$) is $\Sigma_2^P$-complete.
Motivation

Many practical *decision* problems can be represented as QBF.

**Smallest MUS problem**
Find a *smallest* unsatisfiable subformula of a CNF formula.

Decision version (for $\varphi$ and $k$) is $\Sigma^P_2$-complete.

**Quantified MaxSAT (QMaxSAT)**
Find a solution of a QBF that has a *minimal* cost.
Motivation

Many practical decision problems can be represented as QBF.

Smallest MUS problem
Find a smallest unsatisfiable subformula of a CNF formula.

Decision version (for $\varphi$ and $k$) is $\Sigma_2^P$-complete.

Quantified MaxSAT (QMaxSAT)
Find a solution of a QBF that has a minimal cost.

Applications — optimization problems with quantified constraints.
Quantified Boolean Formula

QBF — a quantified generalization of SAT:
Quantified Boolean Formula

QBF — a quantified generalization of SAT:

\[ Q_1X_1...Q_kX_k. \varphi, \text{ where } Q_i \in \{\exists, \forall\} \]
Quantified Boolean Formula

QBF — a quantified generalization of SAT:

- $Q_1X_1\ldots Q_kX_k. \varphi$, where $Q_i \in \{\exists, \forall\}$

- $\overrightarrow{Q}. \varphi$, where $\overrightarrow{Q} = (Q_1X_1\ldots Q_kX_k)$

# short form
Quantified Boolean Formula

QBF — a quantified generalization of SAT:

- $Q_1X_1\ldots Q_kX_k. \varphi$, where $Q_i \in \{\exists, \forall\}$

- $\overrightarrow{Q}. \varphi$, where $\overrightarrow{Q} = (Q_1X_1\ldots Q_kX_k)$  

Example

$\xi = \exists e_1, e_2 \forall x_1, x_2. (\neg e_1 \land \neg e_2) \rightarrow (x_1 \lor x_2)$
Quantified MaxSAT

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

**Definition**

Assignment $A_E$ is a *solution* of $\psi$ iff $\overrightarrow{Q}. \varphi|_{A_E}$ is true.

$M(\psi)$ — set of all solutions of $\psi$. 
Quantified MaxSAT

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

**Definition**

Assignment \( A_E \) is a **solution** of \( \psi \) iff \( \varphi \big|_{A_E} \) is true.

\( M(\psi) \) — set of all solutions of \( \psi \).

**Example**

\[ \xi = \exists e_1, e_2 \ \forall x_1, x_2. (\neg e_1 \land \neg e_2) \rightarrow (x_1 \lor x_2) \]

\( M(\xi) = \{(0,1), (1,0), (1,1)\} \)
Quantified MaxSAT

\[ \psi = \exists E \vec{Q}. \varphi \]

**Definition**

Assignment \( A_E \) is a *solution* of \( \psi \) iff \( \vec{Q}. \varphi \mid_{A_E} \) is true.

\( \mathcal{M}(\psi) \) — set of all solutions of \( \psi \).

**Example**

\[ \xi = \exists e_1, e_2 \ \forall x_1, x_2. \ (\neg e_1 \land \neg e_2) \rightarrow (x_1 \lor x_2) \]

\( \mathcal{M}(\xi) = \{(0, 1), (1, 0), (1, 1)\} \)

What solution is the best?
Quantified MaxSAT

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

**Definition**

Assignment \( A_E \) is a **solution** of \( \psi \) iff \( \overrightarrow{Q}. \varphi \mid_{A_E} \) is true.

\( M(\psi) \) — set of all solutions of \( \psi \).

**Example**

\[ \xi = \exists e_1, e_2 \ \forall x_1, x_2. \ (\neg e_1 \land \neg e_2) \rightarrow (x_1 \lor x_2) \]

\( M(\xi) = \{(0,1), (1,0), (1,1)\} \)

What solution is the best?

**QMaxSAT**

Consider a **cost function** \( f(e_1, \ldots, e_l) = \sum_{i=1}^{l} a_i \cdot e_i, |E| = l \).

**Find** \( A_E \in M(\psi) \) s. t. \( \forall B_E \in M(\psi): f(A_E) \leq f(B_E) \).
Quantified MaxSAT

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

**Definition**

Assignment \( A_E \) is a **solution** of \( \psi \) iff \( \overrightarrow{Q}. \varphi|_{A_E} \) is true.

\( M(\psi) \) — set of all solutions of \( \psi \).

**Example**

\[ \xi = \exists e_1, e_2 \ \forall x_1, x_2. \ (\neg e_1 \land \neg e_2) \rightarrow (x_1 \lor x_2) \]

\( M(\xi) = \{ (0, 1), (1, 0), (1, 1) \} \)

**What solution is the best?**

\[ f(e_1, e_2) = 2 \cdot e_1 + 3 \cdot e_2 \]

**QMaxSAT**

Consider a **cost function** \( f(e_1, \ldots, e_l) = \sum_{i=1}^{l} a_i \cdot e_i, |E| = l. \)

**Find** \( A_E \in M(\psi) \) s. t. \( \forall B_E \in M(\psi): f(A_E) \leq f(B_E) \).
Quantified MaxSAT

\[ \psi = \exists E \overline{Q}. \varphi \]

**Definition**

Assignment \( A_E \) is a **solution** of \( \psi \) iff \( \overline{Q}. \varphi |_{A_E} \) is true.

\( \mathcal{M}(\psi) \) — set of all solutions of \( \psi \).

**Example**

\[ \xi = \exists e_1, e_2 \ \forall x_1, x_2. \ (\neg e_1 \land \neg e_2) \rightarrow (x_1 \lor x_2) \]

\[ \mathcal{M}(\xi) = \{(0, 1), (1, 0), (1, 1)\} \]

**What solution is the best?**

\[ f(e_1, e_2) = 2 \cdot e_1 + 3 \cdot e_2 \quad f(1, 0) = \min_{\mathcal{M}(\xi)} f(e_1, e_2) = 2 \]

**QMaxSAT**

Consider a cost function \( f(e_1, \ldots, e_l) = \sum_{i=1}^{l} a_i \cdot e_i, |E| = l \).

**Find** \( A_E \in \mathcal{M}(\psi) \) s.t. \( \forall B_E \in \mathcal{M}(\psi): f(A_E) \leq f(B_E) \).
Approaches

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

\[ \psi = \exists \overrightarrow{Q}. \varphi \]

- **Linear search:**
  \[ \exists \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]
Approaches

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

- Linear search: \[ \exists E \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

Refine LB

\[ \text{LB}_0 \quad \text{OPT} \quad \text{UB} \]
Approaches

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

- Linear search:
  \[ \exists E \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

Refine LB

\[ \text{LB}_0 \quad \text{LB}_1 \quad \text{OPT} \quad \text{UB} \]
Approaches

\[ \psi = \exists E \vec{Q}. \varphi \]

- Linear search:
  \[ \exists E \vec{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

Refine LB

\[ \text{LB}_0 \quad \text{LB}_1 \quad \text{LB}_2 \quad \text{OPT} \quad \text{UB} \]
**Approaches**

\[ \psi = \exists \overrightarrow{Q}. \varphi \]

- **Linear search:**
  \[ \exists \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_L) \leq k) \]

![Diagram showing the search space with labels: Refine LB, LB0, LB1, LB2, LBi, OPT, and UB.](image-url)
Approaches

\[
\psi = \exists E \quad \overrightarrow{Q}. \quad \varphi
\]

- **Linear search:**
  \[
  \exists E \quad \overrightarrow{Q}. \quad \varphi \land (f(e_1, \ldots, e_l) \leq k)
  \]
Approaches

\[ \psi = \exists E \vec{Q}. \varphi \]

- **Linear search:**
  \[ \exists E \vec{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]
Approaches

\[ \psi = \exists \overrightarrow{Q}. \varphi \]

- **Linear search:**
  \[ \exists \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

Refine LB

\[ \text{LB}_0 \quad \text{LB}_1 \quad \text{LB}_2 \quad \text{LB}_i \quad \text{UB} \]

Refine UB
Approaches

\[ \psi = \exists \overrightarrow{Q}. \varphi \]

- Linear search:
  \[ \exists \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

Refine LB
\[ \text{LB}_0 \quad \text{LB}_1 \quad \text{LB}_2 \quad \text{LB}_i \quad \text{UB} \]

OPT
\[ \text{LB} \quad \text{UB}_i \quad \text{UB}_2 \quad \text{UB}_1 \quad \text{UB}_o \]

Refine UB
Approaches

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

- **Linear search:**
  \[ \exists E \overrightarrow{Q}. \varphi \wedge (f(e_1, \ldots, e_l) \leq k) \]

- **Core-guided search:**

![Diagram of search approaches with bounds and decision points]
Approaches

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

- **Linear search:**
  \[ \exists E \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

- **Core-guided search:**
  \[ f(e_1, \ldots, e_l) = \sum_{i=1}^{l} e_l \]
  # unweighted QMaxSAT
Approaches

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

- **Linear search:**
  \[ \exists E \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

  
  Refine LB
  \[ \text{LB}_0 \quad \text{LB}_1 \quad \text{LB}_2 \quad \text{LB}_i \quad \text{UB} \]
  
  OPT
  \[ \text{OPT} \quad \text{OPT} \quad \text{OPT} \quad \text{OPT} \] 
  
  Refine UB
  \[ \text{UB}_0 \quad \text{UB}_1 \quad \text{UB}_2 \quad \text{UB}_i \quad \text{UB} \]

- **Core-guided search:**
  
  \[ f(e_1, \ldots, e_l) = \sum_{i=1}^{l} e_i \]  
  # unweighted QMaxSAT

  \[ \varphi_S = \{ \neg e_1, \ldots, \neg e_l \} \]  
  # soft clauses
Approaches

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

- **Linear search:**
  \[ \exists E \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

- **Core-guided search:**
  - \( f(e_1, \ldots, e_l) = \sum_{i=1}^{l} e_l \) (unweighted QMaxSAT)
  - \( \varphi_S = \{\neg e_1, \ldots, \neg e_l\} \) (soft clauses)
  - \( \#(\varphi_S, A_E) = \sum_{c \in \varphi_S} c|_{A_E} \) (number of satisfied soft clauses)
Approaches

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

- Linear search:
  \[ \exists E \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

- Core-guided search:
  \[ f(e_1, \ldots, e_l) = \sum_{i=1}^{l} e_l \]
  \[ \varphi_S = \{\neg e_1, \ldots, \neg e_l\} \]
  \[ #(\varphi_S, \mathcal{A}_E) = \sum_{c \in \varphi_S} c|_{\mathcal{A}_E} \]
  \[ \forall \mathcal{A}_E: \]
  \[ f(\mathcal{A}_E) = y \iff #(\varphi_S, \mathcal{A}_E) = l - y \]
Approaches

\[ \psi = \exists E \vec{Q}. \varphi \]

- **Linear search:**
  \[ \exists E \vec{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

- **Core-guided search:**
  \[ f(e_1, \ldots, e_l) = \sum_{i=1}^{l} e_l \]
  \[ \varphi_S = \{ \neg e_1, \ldots, \neg e_l \} \]
  \[ #(\varphi_S, A_E) = \sum_{c \in \varphi_S} c |_{A_E} \]
  \[ \forall A_E: \]
  \[ f(A_E) = y \iff #(\varphi_S, A_E) = l - y \]

\[ \psi' = \exists E \vec{Q}. \varphi \land \varphi_S \]

# QBF to decide iteratively
Approaches

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

- **Linear search:**
  \[ \exists E \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

- **Core-guided search:**
  - \( f(e_1, \ldots, e_l) = \sum_{i=1}^{l} e_l \)  \# unweighted QMaxSAT
  - \( \varphi_S = \{-e_1, \ldots, -e_l\} \)  \# soft clauses
  - \( \#(\varphi_S, A_E) = \sum_{c \in \varphi_S} c|_{A_E} \)  \# number of satisfied soft clauses
  - \( \forall A_E: \)
    \[ f(A_E) = y \iff \#(\varphi_S, A_E) = l - y \]

- \( \psi' = \exists E \overrightarrow{Q}. \varphi \land \varphi_S \)  \# QBF to decide iteratively
- \textbf{find} \( A_E \in \mathcal{M}(\psi) \) that maximizes \( \#(\varphi_S, A_E) \)
\section*{Approaches}

\[ \psi = \exists E \overrightarrow{Q}. \varphi \]

- **Linear search:**
  \[ \exists E \overrightarrow{Q}. \varphi \land (f(e_1, \ldots, e_l) \leq k) \]

- **Core-guided search:**
  \begin{itemize}
  \item \[ f(e_1, \ldots, e_l) = \sum_{i=1}^{l} e_i \]  
    \# unweighted QMaxSAT
  \item \[ \varphi_S = \{ \neg e_1, \ldots, \neg e_l \} \]  
    \# soft clauses
  \item \[ \#(\varphi_S, A_E) = \sum_{c \in \varphi_S} c|_{A_E} \]  
    \# number of satisfied soft clauses
  \item \[ \forall A_E: \quad f(A_E) = y \iff \#(\varphi_S, A_E) = l - y \]
  \end{itemize}

- \[ \psi' = \exists E \overrightarrow{Q}. \varphi \land \varphi_S \]  
  \# QBF to decide iteratively

**Definition**

Formula \( \varphi_C = \varphi \land \varphi'_S, \varphi'_S \subseteq \varphi_S \), is an **unsatisfiable core** of \( \psi' \) if \( \exists E \overrightarrow{Q}. \varphi_C \) is false.
Algorithm QMSU₁

• Based on the Fu&Malik’s algorithm (a.k.a. MSU₁ or WPM₁)
Algorithm QMSU₁

- Based on the Fu&Malik’s algorithm (a.k.a. MSU₁ or WPM₁)

Differences:
- **QBF** oracle instead of SAT oracle
- hard part *can be* in non-CNF
Algorithm QMSU₁

- Based on the **Fu&Malik**’s algorithm (a.k.a. MSU₁ or WPM₁)

**Differences:**
- **QBF** oracle instead of SAT oracle
- hard part *can be* in non-CNF

**Basic principles:**

1. $\psi'_R \leftarrow \psi' = \exists E \overrightarrow{Q}. \varphi \land \varphi_S$
Algorithm QMSU₁

Based on the **Fu&Malik**’s algorithm (a.k.a. **MSU₁** or **WPM₁**)

**Differences:**
- **QBF** oracle instead of SAT oracle
- hard part *can be* in non-CNF

**Basic principles:**

1. $ψ'_R ← ψ' = ∃E Q. ϕ ∧ ϕ_S$
2. **while** $ψ'_R$ is false:  
   # ask a QBF oracle
Algorithm QMSU₁

- Based on the **Fu&Malik**’s algorithm (a.k.a. MSU₁ or WPM₁)

  Differences:
  - **QBF** oracle instead of SAT oracle
  - hard part *can be* in non-CNF

- Basic principles:

  1. \( \psi'_R \leftarrow \psi' = \exists E \overrightarrow{Q}. \varphi \land \varphi_S \)
  2. **while** \( \psi'_R \) is false:
  3. extract unsatisfiable core \( \varphi_C \)

# ask a QBF oracle
Algorithm QMSU₁

- Based on the **Fu&Malik**’s algorithm (a.k.a. MSU₁ or WPM₁)

**Differences:**
- **QBF** oracle instead of SAT oracle
- hard part *can be* in non-CNF

**Basic principles:**

1. \( \psi'_R \leftarrow \psi' = \exists E \bar{Q}. \varphi \land \varphi_S \)
2. **while** \( \psi'_R \) is false:
   - extract unsatisfiable core \( \varphi_C \)
   - relax soft part of \( \varphi_C \)

# ask a QBF oracle
Algorithm QMSU₁

- Based on the Fu&Malik’s algorithm (a.k.a. MSU₁ or WPM₁)

Differences:
- QBF oracle instead of SAT oracle
- hard part can be in non-CNF

Basic principles:

1. $ψ'_R ← ψ' = \exists E \overrightarrow{Q}. ϕ \land ϕ_S$
2. while $ψ'_R$ is false: # ask a QBF oracle
3. extract unsatisfiable core $ϕ_C$
4. relax soft part of $ϕ_C$
5. update $ψ'_R$
Algorithm QMSU1

input : $\psi = \exists E \vec{Q}. \varphi$, and $\varphi_S$

1. $R_{all} \leftarrow \emptyset$  
   # set of all relaxation variables
2. while true:
3.   $\psi'_R = \exists E \exists R_{all} \vec{Q}. \varphi \land \varphi_S$
4.   $(st, \varphi_C, A_E) \leftarrow \text{QBF}(\psi'_R)$  
   # calling a QBF oracle
5.   if $st = true$:
6.     return $A_E$
7.     $R \leftarrow \emptyset$
8.     foreach $c \in \text{Soft}(\varphi_C)$:
9.       let $r$ be a new relaxation variable
10.      $R \leftarrow R \cup \{r\}$
11.      $\varphi_S \leftarrow \varphi_S \setminus \{c\} \cup \{c \lor r\}$
12.      $\varphi \leftarrow \varphi \land \text{CNF}(\sum_{r \in R} r \leq 1)$  
   # updating the hard part
13.     $R_{all} \leftarrow R_{all} \cup R$  
   # relaxing the core
Algorithm QMSU₁

input : $\psi = \exists E \vec{Q}. \varphi$, and $\varphi_S$

1. $R_{all} \leftarrow \emptyset$  
   # set of all relaxation variables
2. while true:
3.     $\psi'_R = \exists E \exists R_{all} \vec{Q}. \varphi \land \varphi_S$
4.     $(st, \varphi_C, \mathcal{A}_E) \leftarrow \text{QBF}(\psi'_R)$
5.     if $st = \text{true}$:
6.         return $\mathcal{A}_E$
7.     $R \leftarrow \emptyset$
8.     foreach $c \in \text{Soft}(\varphi_C)$:
9.         let $r$ be a new relaxation variable
10.        $R \leftarrow R \cup \{r\}$
11.       $\varphi_S \leftarrow \varphi_S \setminus \{c\} \cup \{c \lor r\}$
12.       $\varphi \leftarrow \varphi \land \text{CNF}(\sum_{r \in R} r \leq 1)$
13.      $R_{all} \leftarrow R_{all} \cup R$
14.      # relaxing the core
15.    $\varphi_S \leftarrow \varphi_S \setminus \{c\} \cup \{c \lor r\}$
16.    $\varphi \leftarrow \varphi \land \text{CNF}(\sum_{r \in R} r \leq 1)$
17.    $R_{all} \leftarrow R_{all} \cup R$
18.    # updating the hard part
19.  # calling a QBF oracle
20.  # setting the relaxation variables
Algorithm QMSU₁

\[ \text{input} : \psi = \exists E \quad Q \quad \varphi, \text{ and } \varphi_S \]

1. \( \mathcal{R}_{\text{all}} \leftarrow \emptyset \)  # set of all relaxation variables
2. \textbf{while} true:
3. \( \psi'_R = \exists E \exists \mathcal{R}_{\text{all}} \quad Q \quad \varphi \land \varphi_S \)
4. \((\text{st, } \varphi_C, \mathcal{A}_E) \leftarrow \text{QBF}(\psi'_R)\)  # calling a QBF oracle
5. \textbf{if} \text{ st } = \text{ true}:
6. \quad \textbf{return} \mathcal{A}_E
7. \mathcal{R} \leftarrow \emptyset
8. \textbf{foreach} \ c \in \text{Soft}(\varphi_C):  # relaxing the core
9. \quad \textbf{let} \ r \text{ be a new relaxation variable}
10. \quad \mathcal{R} \leftarrow \mathcal{R} \cup \{r\}
11. \quad \varphi_S \leftarrow \varphi_S \setminus \{c\} \cup \{c \lor r\}
12. \varphi \leftarrow \varphi \land \text{CNF}(\sum_{r \in \mathcal{R}} r \leq 1)  # updating the hard part
13. \mathcal{R}_{\text{all}} \leftarrow \mathcal{R}_{\text{all}} \cup \mathcal{R}
CEGAR-based 2QBF

\[ \exists X \forall Y. \ \varphi_H \land \varphi_S \]
CEGAR-based 2QBF

\[ \exists X \forall Y. \, \varphi_H \land \varphi_S \]

\[ \Leftrightarrow \]

Full expansion:

\[ \exists X. \, \bigwedge_{\nu \in \{0, 1\}^{|Y|}} (\varphi_H \land \varphi_S) \big|_{\nu} \]
CEGAR-based 2QBF

\[ \exists X \forall Y. \varphi_H \land \varphi_S \]

\[ \iff \]

**Full expansion:**

\[ \exists X. \bigwedge_{\nu \in \{0, 1\}^{|Y|}} (\varphi_H \land \varphi_S)_{\nu} \]

**Partial expansion:**

\[ \exists X. \bigwedge_{\nu \in W} (\varphi_H \land \varphi_S)_{\nu} \quad W \subseteq \{0, 1\}^{|Y|} \]
CEGAR-based 2QBF

\[ \exists X \forall Y. \ \varphi_H \land \varphi_S \]

\[ \iff \]

**Full expansion:**

\[ \exists X. \bigwedge_{\nu \in \{0,1\}^{|Y|}} (\varphi_H \land \varphi_S) \bigg|_\nu \]

**Partial expansion:**

\[ \exists X. \bigwedge_{\nu \in W} (\varphi_H \land \varphi_S) \bigg|_\nu \quad W \subseteq \{0,1\}^{|Y|} \]

Gradual strengthening of abstractions until a solution is found.
CEGAR-based 2QBF

$$\exists X \forall Y. \varphi_H \land \varphi_S$$

$$\Leftrightarrow$$

Full expansion:

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

Partial expansion:

$$\exists X. \bigwedge_{\nu \in W} (\varphi_H \land \varphi_S) \big|_{\nu} \quad W \subseteq \{0, 1\}^{\|Y\|}$$

Gradual strengthening of abstractions until a solution is found
CEGAR-based 2QBF

$$\exists X \forall Y. \varphi_H \land \varphi_S$$

$$\Leftrightarrow$$

Full expansion:

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

Partial expansion:

$$\exists X. \bigwedge_{\nu \in \mathcal{W}} (\varphi_H \land \varphi_S)\big|_{\nu} \quad \mathcal{W} \subseteq \{0,1\}^{|Y|}$$

Gradual strengthening of abstractions until a solution is found
CEGAR-based 2QBF

\[ \exists X \forall Y. \varphi_H \land \varphi_S \]

\[ \iff \]

**Full expansion:**

\[ \exists X. \bigwedge_{\nu \in \{0,1\}^{|Y|}} (\varphi_H \land \varphi_S) \big|_\nu \]

**Partial expansion:**

\[ \exists X. \bigwedge_{\nu \in W} (\varphi_H \land \varphi_S) \big|_\nu \quad W \subseteq \{0,1\}^{|Y|} \]

Gradual strengthening of abstractions until a solution is found
CEGAR-based 2QBF

\[ \exists X \forall Y. \; \varphi_H \land \varphi_S \]

\[ \Leftrightarrow \]

**Full expansion:**

\[ \exists X. \bigwedge_{\nu \in \{0,1\}^{|Y|}} (\varphi_H \land \varphi_S)\bigg|_{\nu} \]

**Partial expansion:**

\[ \exists X. \bigwedge_{\nu \in W} (\varphi_H \land \varphi_S)\bigg|_{\nu} \quad W \subseteq \{0,1\}^{|Y|} \]

Gradual strengthening of abstractions until a solution is found
CEGAR-based 2QBF

\[ \exists X \forall Y. \varphi_H \land \varphi_S \]

\[ \iff \]

**Full expansion:**
\[ \exists X. \bigwedge_{\nu \in \{0,1\}^{|Y|}} (\varphi_H \land \varphi_S)_{\nu} \]

**Partial expansion:**
\[ \exists X. \bigwedge_{\nu \in W} (\varphi_H \land \varphi_S)_{\nu} \quad W \subseteq \{0, 1\}^{|Y|} \]

Gradual strengthening of abstractions until a solution is found
Computing Cores in CEGAR-based 2QBF

**input**  : $\exists X \forall Y. \varphi_H \land \varphi_S$

1. $\omega \leftarrow \emptyset$
2. **while** true:
3.   $\varphi \leftarrow \text{CNF}\left( \bigwedge_{\nu \in \omega} \varphi_H |_{\nu} \right) \cup \bigwedge_{\nu \in \omega} \varphi_S |_{\nu}$
4.   $(\text{st}_1, \mu, \varphi_C) \leftarrow \text{SAT}(\varphi)$  
   # candidate
5.   **if** $\text{st}_1 = \text{false}$:
6.      $\varphi'_S \leftarrow \{ c \in \varphi_S | c' \in \varphi_C, \nu \in \omega, c' = c |_{\nu} \}$
7.      **return** (false, $\varphi_H \land \varphi'_S$)  
    # no candidate found
8.   $(\text{st}_2, \nu) \leftarrow \text{SAT}\left( \neg(\varphi_H \land \varphi_S) |_{\mu} \right)$  
    # counterexample
9.   **if** $\text{st}_2 = \text{false}$:
10.      **return** (true, $\mu$)  
    # solution found
11.   $\omega \leftarrow \omega \cup \{ \nu \}$
Computing Cores in CEGAR-based 2QBF

\textbf{input} : \exists X \forall Y. \varphi_H \land \varphi_S

\begin{algorithmic}
\State $\omega \leftarrow \emptyset$
\While {true:}
\State $\varphi \leftarrow \text{CNF}\left(\bigwedge_{\nu \in \omega} \varphi_H|_{\nu}\right) \cup \bigwedge_{\nu \in \omega} \varphi_S|_{\nu}$
\State $(\text{st}_1, \mu, \varphi_C) \leftarrow \text{SAT}(\varphi)$ \hfill \# candidate
\If {st$_1$ = false:}
\State $\varphi'_S \leftarrow \{c \in \varphi_S \mid c' \in \varphi_C, \nu \in \omega, c' = c|_{\nu}\}$
\State \textbf{return} (false, $\varphi_H \land \varphi'_S$) \hfill \# no candidate found
\EndIf
\State $(\text{st}_2, \nu) \leftarrow \text{SAT}\left(\neg(\varphi_H \land \varphi_S)|_{\mu}\right)$ \hfill \# counterexample
\If {st$_2$ = false:}
\State \textbf{return} (true, $\mu$) \hfill \# solution found
\EndIf
\State $\omega \leftarrow \omega \cup \{\nu\}$
\EndWhile
\end{algorithmic}
Computing Cores in CEGAR-based 2QBF

**input** : $\exists X \forall Y. \varphi_H \land \varphi_S$

1. $\omega \leftarrow \emptyset$
2. **while** true:
   1. $\varphi \leftarrow \text{CNF}(\bigwedge_{\nu \in \omega} \varphi_H|_{\nu}) \cup \bigwedge_{\nu \in \omega} \varphi_S|_{\nu}$
   2. $(\text{st}_1, \mu, \varphi_C) \leftarrow \text{SAT}(\varphi)$
      
        # candidate

5. **if** $\text{st}_1 = \text{false}$:
   1. $\varphi'_S \leftarrow \{c \in \varphi_S | c' \in \varphi_C, \nu \in \omega, c' = c|_{\nu}\}$
   2. **return** (false, $\varphi_H \land \varphi'_S$)
      
        # no candidate found

8. $(\text{st}_2, \nu) \leftarrow \text{SAT}(\neg(\varphi_H \land \varphi_S)|_{\mu})$

   # counterexample

9. **if** $\text{st}_2 = \text{false}$:
   1. **return** (true, $\mu$)
      
        # solution found

11. $\omega \leftarrow \omega \cup \{\nu\}$
Smallest MUS Problem

Definition

Formula $\psi^*$, $\psi^* \subseteq \varphi$, is called a **smallest MUS** of $\varphi$ if

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

Example

$\varphi = \{ x_2 \lor \neg x_3 \lor \neg x_4, x_1 \lor x_2, x_3, \neg x_1, x_4, \neg x_2 \}$

$\varphi$ has 2 MUSes:
Smallest MUS Problem

Definition

Formula $\psi^*,$ $\psi^* \subseteq \phi,$ is called a smallest MUS of $\phi$ if

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

Example

$\phi = \{ x_2 \lor \neg x_3 \lor \neg x_4, x_1 \lor x_2, x_3, \neg x_1, x_4, \neg x_2 \}$

$\phi$ has 2 MUSes:

- $|\psi_1| = 4$
Smallest MUS Problem

Definition

Formula $\psi^*$, $\psi^* \subseteq \varphi$, is called a **smallest MUS** of $\varphi$ if

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

Example

$\varphi = \{ x_2 \lor \neg x_3 \lor \neg x_4, x_1 \lor x_2, x_3, \neg x_1, x_4, \neg x_2 \}$

$\varphi$ has 2 MUSes:

- $|\psi_1| = 4$
- $|\psi_2| = 3$
Smallest MUS Problem

Definition

Formula $\psi^*, \psi^* \subseteq \varphi$, is called a **smallest MUS** of $\varphi$ if

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

Example

$\varphi = \{ x_2 \lor \neg x_3 \lor \neg x_4, x_1 \lor x_2, x_3, \neg x_1, x_4, \neg x_2 \}$

$\varphi$ has 2 MUSes:

- $|\psi_1| = 4$
- $|\psi_2| = 3 < |\psi_1| \Rightarrow \psi_2 = \psi^*$
SMUS as QMaxSAT

- Original formula $\varphi = \{c_1, \ldots, c_m\}$
SMUS as QMaxSAT

- Original formula $\varphi = \{c_1, \ldots, c_m\}$

- QBF formulation of SMUS:
SMUS as QMaxSAT

- **Original formula** $\varphi = \{c_1, \ldots, c_m\}$

- **QBF formulation of SMUS:**
  - $S = \{s_1, \ldots, s_m\}$ # selection variables
  - $\varphi_R = \{c_1 \lor \neg s_1, \ldots, c_m \lor \neg s_m\}$ # extended formula
SMUS as QMaxSAT

- Original formula $\varphi = \{c_1, \ldots, c_m\}$

- QBF formulation of SMUS:
  - $S = \{s_1, \ldots, s_m\}$  
    # selection variables
  - $\varphi_R = \{c_1 \lor \neg s_1, \ldots, c_m \lor \neg s_m\}$  
    # extended formula
  - $\varphi_{\text{unsat}} = \exists S \forall X. \neg \varphi_R$  
    # QBF for finding UNSAT subformula
SMUS as QMaxSAT

- Original formula $\varphi = \{c_1, \ldots, c_m\}$

- QBF formulation of SMUS:
  - $S = \{s_1, \ldots, s_m\}$ # selection variables
  - $\varphi_R = \{c_1 \lor \neg s_1, \ldots, c_m \lor \neg s_m\}$ # extended formula
  - $\varphi_{unsat} = \exists S \forall X. \neg \varphi_R$ # QBF for finding UNSAT subformula
  - $f(s_1, \ldots, s_m) = \sum_{i=1}^{m} s_i$ # objective function
SMUS as QMaxSAT

- Original formula \( \varphi = \{c_1, \ldots, c_m\} \)

- QBF formulation of SMUS:
  - \( S = \{s_1, \ldots, s_m\} \)  
    
    # selection variables
  - \( \varphi_R = \{c_1 \lor \neg s_1, \ldots, c_m \lor \neg s_m\} \)  
    
    # extended formula
  - \( \varphi_{\text{unsat}} = \exists S \forall X. \neg \varphi_R \)  
    
    # QBF for finding UNSAT subformula
  - \( f(s_1, \ldots, s_m) = \sum_{i=1}^{m} s_i \)  
    
    # objective function

- \text{find} \( A_S \in M(\varphi_{\text{unsat}}) \) s.t. \( \forall B_S \in M(\varphi_{\text{unsat}}): f(A_S) \leq f(B_S) \)
SMUS as QMaxSAT

- Original formula $\varphi = \{c_1, \ldots, c_m\}$

- QBF formulation of SMUS:
  - $S = \{s_1, \ldots, s_m\}$ # selection variables
  - $\varphi_R = \{c_1 \lor \neg s_1, \ldots, c_m \lor \neg s_m\}$ # extended formula
  - $\varphi_{unsat} = \exists S \forall X. \neg \varphi_R$ # QBF for finding UNSAT subformula
  - $f(s_1, \ldots, s_m) = \sum_{i=1}^{m} s_i$ # objective function

  - **find** $A_S \in M(\varphi_{unsat})$ s.t. $\forall B_S \in M(\varphi_{unsat})$: $f(A_S) \leq f(B_S)$

- Result QBF to decide iteratively:
  - $\exists S \forall X. \neg \varphi_R \land (f(s_1, \ldots, s_m) \leq k)$ # linear search
SMUS as QMaxSAT

- Original formula $\varphi = \{c_1, \ldots, c_m\}$

- QBF formulation of SMUS:
  - $S = \{s_1, \ldots, s_m\}$  # selection variables
  - $\varphi_R = \{c_1 \lor \neg s_1, \ldots, c_m \lor \neg s_m\}$  # extended formula
  - $\varphi_{\text{unsat}} = \exists S \forall X. \neg \varphi_R$  # QBF for finding UNSAT subformula
  - $f(s_1, \ldots, s_m) = \sum_{i=1}^{m} s_i$  # objective function

- **find** $A_S \in M(\varphi_{\text{unsat}})$ s.t. $\forall B_S \in M(\varphi_{\text{unsat}}): f(A_S) \leq f(B_S)$

- Result QBF to decide iteratively:
  - $\exists S \forall X. \neg \varphi_R \land (f(s_1, \ldots, s_m) \leq k)$  # linear search
  - $\exists S \forall X. \neg \varphi_R \land \varphi_S$, where $\varphi_S = \{\neg s_1, \ldots, \neg s_m\}$  # core-guided search  # soft constraints
Improvements of the Approach

*Digger* — state of the art for SMUS.
Improvements of the Approach

**Digger** — state of the art for SMUS.

Disjoint MCS enumeration can help
Improvements of the Approach

*Digger* — state of the art for SMUS.

Disjoint MCS enumeration can help — it improves the lower bound.
Improvements of the Approach

*Digger* — state of the art for SMUS.

Disjoint MCS enumeration can help — it improves the lower bound.

**Definition**

Formula $\psi$, $\psi \subseteq \varphi$, is called a *minimal correction set* of $\varphi$ if

1. $\varphi$ is unsatisfiable
2. $\varphi \setminus \psi$ is satisfiable
3. for any clause $c \in \psi$: $\varphi \setminus \psi \cup \{c\}$ is unsatisfiable.
Improvements of the Approach

*Digger* — state of the art for SMUS.

Disjoint MCS enumeration can help — it improves the lower bound.

**Definition**

Formula $\psi$, $\psi \subseteq \varphi$, is called a *minimal correction set* of $\varphi$ if

1. $\varphi$ is unsatisfiable
2. $\varphi \setminus \psi$ is satisfiable
3. for any clause $c \in \psi$: $\varphi \setminus \psi \cup \{c\}$ is unsatisfiable.

**Fact**

*Any* MUS of $\varphi$ is a *minimal hitting set* of the complete set of MCSes of $\varphi$. 
Improvements of the Approach

*Digger* — state of the art for SMUS.

Disjoint MCS enumeration can help — it improves the lower bound.

**Definition**

Formula $\psi$, $\psi \subseteq \varphi$, is called a *minimal correction set* of $\varphi$ if

1. $\varphi$ is unsatisfiable
2. $\varphi \setminus \psi$ is satisfiable
3. for any clause $c \in \psi$: $\varphi \setminus \psi \cup \{c\}$ is unsatisfiable.

**Fact**

*Any* MUS of $\varphi$ is a *minimal hitting set* of the complete set of MCSes of $\varphi$.

Core-guided search *cannot* use lower bounds, but
Improvements of the Approach

**Digger** — state of the art for SMUS.

Disjoint MCS enumeration can help — it improves the lower bound.

**Definition**

Formula $\psi$, $\psi \subseteq \varphi$, is called a *minimal correction set* of $\varphi$ if

1. $\varphi$ is unsatisfiable
2. $\varphi \setminus \psi$ is satisfiable
3. for any clause $c \in \psi$: $\varphi \setminus \psi \cup \{c\}$ is unsatisfiable.

**Fact**

**Any** MUS of $\varphi$ is a *minimal hitting set* of the complete set of MCSes of $\varphi$.

Core-guided search *cannot* use lower bounds, but

- find *unit* MCSes
Improvements of the Approach

*Digger* — state of the art for SMUS.

Disjoint MCS enumeration can help — it improves the lower bound.

**Definition**

Formula $\psi$, $\psi \subseteq \varphi$, is called a *minimal correction set* of $\varphi$ if

1. $\varphi$ is unsatisfiable
2. $\varphi \setminus \psi$ is satisfiable
3. for any clause $c \in \psi$: $\varphi \setminus \psi \cup \{c\}$ is unsatisfiable.

**Fact**

*Any* MUS of $\varphi$ is a *minimal hitting set* of the complete set of MCSes of $\varphi$.

Core-guided search *cannot* use lower bounds, but

- find *unit* MCSes $\Rightarrow$ SMUS contains all of them
Improvements of the Approach

**Digger** — state of the art for SMUS.

Disjoint MCS enumeration can help — it improves the lower bound.

**Definition**

Formula \( \psi, \psi \subseteq \varphi \), is called a **minimal correction set** of \( \varphi \) if

1. \( \varphi \) is unsatisfiable
2. \( \varphi \setminus \psi \) is satisfiable
3. for any clause \( c \in \psi \): \( \varphi \setminus \psi \cup \{c\} \) is unsatisfiable.

**Fact**

**Any** MUS of \( \varphi \) is a **minimal hitting set** of the complete set of MCSes of \( \varphi \).

Core-guided search **cannot** use lower bounds, but

- find **unit** MCSes \( \Rightarrow \) SMUS contains all of them
- **any** MCS is an unsatisfiable core of \( \exists S \forall X. \neg \varphi_R \land \varphi_S \)

# see the paper
Performance Comparison: MinUC vs Digger
Performance Comparison: Linear Search vs Core-Guided

(a) MinUC vs MinUC-LB

(b) MinUC vs MinUC-UB
Number of Solved Instances

MinUC-d vs Digger

Only Digger: 1
Both: 363
Only MinUC-d: 33

MinUC vs Digger

Only Digger: 0
Both: 364
Only MinUC: 80

Total number of instances — 682.
Summary and Future Work

- Novel core-guided algorithm for QMaxSAT
Summary and Future Work

- Novel core-guided algorithm for QMaxSAT
- Generation of unsatisfiable cores with CEGAR-based QBF
Summary and Future Work

- Novel core-guided algorithm for QMaxSAT
- Generation of unsatisfiable cores with CEGAR-based QBF
- Smallest MUS solved as QMaxSAT
Summary and Future Work

- Novel core-guided algorithm for QMaxSAT
- Generation of unsatisfiable cores with CEGAR-based QBF
- Smallest MUS solved as QMaxSAT

- Other quantified optimization problems
Summary and Future Work

- Novel core-guided algorithm for QMaxSAT
- Generation of unsatisfiable cores with CEGAR-based QBF
- Smallest MUS solved as QMaxSAT

- Other quantified optimization problems
- More core-guided algorithms for QMaxSAT
Summary and Future Work

- Novel core-guided algorithm for QMaxSAT
- Generation of unsatisfiable cores with CEGAR-based QBF
- Smallest MUS solved as QMaxSAT

- Other quantified optimization problems
- More core-guided algorithms for QMaxSAT
- Integration of DPLL-based QBF solvers
Summary and Future Work

- Novel core-guided algorithm for QMaxSAT
- Generation of unsatisfiable cores with CEGAR-based QBF
- Smallest MUS solved as QMaxSAT

- Other quantified optimization problems
- More core-guided algorithms for QMaxSAT
- Integration of DPLL-based QBF solvers
- CEGAR-based vs DPLL-based comparison (unsatisfiable cores)
Thank you for your attention!