Boolean-Based Optimization
Algorithms & Applications

Joao Marques-Silva

School of Computer Science and Informatics
University College Dublin

MSR Software Sumit, Paris, France, April 2011
Why Optimization? Software Package Upgrades

[Adapted from Argelich&Lynce'08]

- Universe of software packages: \( \{p_1, \ldots, p_n\} \)
- Associate \( x_i \) with \( p_i \): \( x_i = 1 \) iff \( p_i \) is installed
- Constraints associated with package \( p_i \): \( (p_i, D_i, C_i) \)
  - \( D_i \): dependencies (required packages) for installing \( p_i \)
  - \( C_i \): conflicts (disallowed packages) for installing \( p_i \)

- Example problem: Maximum Installability
  - Maximum number of packages that can be installed

Package constraints:

\[
(p_1, \{p_2 \lor p_3\}, \{p_4\})
(p_2, \{p_3\}, \{p_4\})
(p_3, \{p_2\}, \emptyset)
(p_4, \{p_2, p_3\}, \emptyset)
\]
Why Optimization? Software Package Upgrades

[Adapted from Argelich&Lynce’08]

- Universe of software packages: \( \{p_1, \ldots, p_n\} \)
- Associate \( x_i \) with \( p_i \): \( x_i = 1 \) iff \( p_i \) is installed
- Constraints associated with package \( p_i \): \( (p_i, D_i, C_i) \)
  - \( D_i \): dependencies (required packages) for installing \( p_i \)
  - \( C_i \): conflicts (disallowed packages) for installing \( p_i \)

- Example problem: Maximum Installability
  - Maximum number of packages that can be installed

Package constraints:

\[
\begin{align*}
(p_1, \{p_2 \lor p_3\}, \{p_4\}) \\
(p_2, \{p_3\}, \{p_4\}) \\
(p_3, \{p_2\}, \emptyset) \\
(p_4, \{p_2, p_3\}, \emptyset)
\end{align*}
\]

Pseudo-Boolean formulation:

\[
\begin{align*}
\text{max} & \quad x_1 + x_2 + x_3 + x_4 \\
\text{s. t.} & \quad (x_1 \rightarrow x_2 \lor x_3) \land (x_1 \rightarrow \neg x_4) \land \\
& \quad (x_2 \rightarrow x_3) \land (x_2 \rightarrow \neg x_4) \land \\
& \quad (x_3 \rightarrow x_2) \land (x_4 \rightarrow x_2) \land (x_4 \rightarrow x_3)
\end{align*}
\]
Outline

Boolean-Based Optimization

Practical Applications

Boolean Optimization Algorithms

CNF Encodings
  - Cardinality Constraints
  - Pseudo-Boolean Constraints

Conclusions & Future Work
Outline

Boolean-Based Optimization

Practical Applications

Boolean Optimization Algorithms

CNF Encodings
  Cardinality Constraints
  Pseudo-Boolean Constraints

Conclusions & Future Work
Boolean-Based Optimization

- Linear optimization over Boolean domains
  - Can be mildly non-linear (e.g. basic Boolean operators)
Boolean-Based Optimization

- Linear optimization over Boolean domains
  - Can be mildly non-linear (e.g. basic Boolean operators)

- Concrete instantiations:
  - Maximum Satisfiability (MaxSAT)
  - Pseudo-Boolean Optimization (PBO, 0-1 ILP)
  - Weighted-Boolean Optimization (WBO)
  - Can map any problem to any other problem

[Larrosa et al.’08; etc.]
Boolean-Based Optimization

- **Linear optimization over Boolean domains**
  - Can be mildly non-linear (e.g. basic Boolean operators)

- **Concrete instantiations:**
  - **Maximum Satisfiability (MaxSAT)**
  - **Pseudo-Boolean Optimization (PBO, 0-1 ILP)**
  - **Weighted-Boolean Optimization (WBO)**
  - Can map any problem to any other problem

- **Related problems:**
  - Optimization in SMT (MaxSMT)
  - Optimization in CSP (Max-CSP, etc.)
  - Integer Linear Programming (ILP)

[See Larrosa et al.'08; etc.]
What is Maximum Satisfiability?

- CNF Formula:

\[
\begin{align*}
x_6 \lor x_2 & \quad \neg x_6 \lor x_2 & \quad \neg x_2 \lor x_1 & \quad \neg x_1 \\
\neg x_6 \lor x_8 & \quad x_6 \lor \neg x_8 & \quad x_2 \lor x_4 & \quad \neg x_4 \lor x_5 \\
x_7 \lor x_5 & \quad \neg x_7 \lor x_5 & \quad \neg x_5 \lor x_3 & \quad \neg x_3
\end{align*}
\]
What is Maximum Satisfiability?

- CNF Formula:
  \[
  \begin{align*}
  x_6 \lor x_2 \\
  \neg x_6 \lor x_2 \\
  x_6 \lor \neg x_8 \\
  \neg x_6 \lor x_8 \\
  x_7 \lor x_5 \\
  \neg x_7 \lor x_5 \\
  \end{align*}
  \]

- Formula is unsatisfiable
- MaxSAT:
  - Find assignment that maximizes number of satisfied clauses
    - For above formula, solution is 10
- Can be used for solving (linear) Boolean optimization problems
- There are a number of variants of MaxSAT
The MaxSAT Problem(s)

- **MaxSAT:**
  - All clauses are *soft*
  - Find assignment that maximizes number of satisfied *soft* clauses
The MaxSAT Problem(s)

- **MaxSAT:**
  - All clauses are *soft*
  - Find assignment that maximizes number of satisfied *soft* clauses

- **Partial MaxSAT:**
  - Hard clauses *must* be satisfied
  - Find assignment that maximizes number of satisfied *soft* clauses
The MaxSAT Problem(s)

- **MaxSAT:**
  - All clauses are soft
  - Find assignment that maximizes number of satisfied soft clauses

- **Partial MaxSAT:**
  - Hard clauses must be satisfied
  - Find assignment that maximizes number of satisfied soft clauses

- **Weighted MaxSAT**
  - Weights associated with clauses
  - Find assignment that maximizes sum of weights of satisfied clauses
The MaxSAT Problem(s)

- **MaxSAT:**
  - All clauses are *soft*
  - Find assignment that maximizes number of satisfied *soft* clauses

- **Partial MaxSAT:**
  - Hard clauses **must** be satisfied
  - Find assignment that maximizes number of satisfied *soft* clauses

- **Weighted MaxSAT**
  - Weights associated with clauses
  - Find assignment that maximizes sum of weights of satisfied clauses

- **Weighted partial MaxSAT**
  - Weights associated with *soft* clauses
  - Hard clauses **must** be satisfied
  - Find assignment that maximizes sum of weights of satisfied *soft* clauses
Pseudo-Boolean Constraints & Optimization

- **Pseudo-Boolean Constraints:**
  - Boolean variables: \( x_1, \ldots, x_n \)
  - Linear inequalities:
    \[
    \sum_{j \in N} a_{ij} l_j \geq b_i, \quad l_j \in \{x_j, \bar{x}_j\}, \quad x_j \in \{0, 1\}, \quad a_{ij}, b_i \in \mathbb{N}_0^+
    \]

- **Algorithms:**
  - Adapt SAT solver and handle constraints natively
  - Encode constraints to clausal form
    [e.g. Manquinho&Marques-Silva’00]
    [e.g. Een&Sörensson’06]

- **Pseudo-Boolean Optimization:**
  \[
  \text{minimize} \quad \sum_{j \in N} c_j \cdot x_j \\
  \text{subject to} \quad \sum_{j \in N} a_{ij} l_j \geq b_i, \\
  \quad l_j \in \{x_j, \bar{x}_j\}, \quad x_j \in \{0, 1\}, \quad a_{ij}, b_i, c_j \in \mathbb{N}_0^+
  \]
The problem:

\[
\begin{align*}
\text{min} & \quad \sum_{j=1}^{n} c_j x_j \\
\text{s. t.} & \quad \sum_{j=1}^{n} a_{i,j} x_j \geq b_i, \quad i = 1, \ldots, m
\end{align*}
\]
Translating PBO to MaxSAT

- The problem:

\[
\begin{align*}
\text{min} & \quad \sum_{j=1}^{n} c_j x_j \\
\text{s. t.} & \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, \ldots, m
\end{align*}
\]

- Weighted partial MaxSAT formulation:
  - Represent pseudo-Boolean constraints as hard clauses
    - Can convert PB constraints to SAT
  - Weighted soft clauses: \((\neg x_j)\) with weight \(c_j\), \(j = 1, \ldots, n\)
    - If negated literals are maximized, then positive literals are minimized
  - If \(c_j = 1\), then instance of partial MaxSAT
Outline

Boolean-Based Optimization

Practical Applications

Boolean Optimization Algorithms

CNF Encodings
  Cardinality Constraints
  Pseudo-Boolean Constraints

Conclusions & Future Work
### Software Package Upgrades with MaxSAT

- **Universe of software packages:** \({\{p_1, \ldots, p_n\}}\)
- **Associate** \(x_i\) **with** \(p_i\): \(x_i = 1\) iff \(p_i\) **is installed**
- **Constraints associated with package** \(p_i\): \((p_i, D_i, C_i)\)
  - \(D_i\): dependencies (required packages) for installing \(p_i\)
  - \(C_i\): conflicts (disallowed packages) for installing \(p_i\)

- **Example problem:** **Maximum Installability**
  - Maximum number of packages that can be installed
  - Package constraints represent **hard** clauses
  - **Soft** clauses: \((x_i)\)

**Package constraints:**

\[
(p_1, \{p_2 \lor p_3\}, \{p_4\})
(p_2, \{p_3\}, \{p_4\})
(p_3, \{p_2\}, \emptyset)
(p_4, \{p_2, p_3\}, \emptyset)
\]
Software Package Upgrades with MaxSAT

- Universe of software packages: \( \{p_1, \ldots, p_n\} \)
- Associate \( x_i \) with \( p_i \): \( x_i = 1 \) iff \( p_i \) is installed
- Constraints associated with package \( p_i \): \((p_i, D_i, C_i)\)
  - \( D_i \): dependencies (required packages) for installing \( p_i \)
  - \( C_i \): conflicts (disallowed packages) for installing \( p_i \)
- Example problem: **Maximum Installability**
  - Maximum number of packages that can be installed
  - Package constraints represent **hard** clauses
  - Soft clauses: \( (x_i) \)

Package constraints:

- \((p_1, \{p_2 \lor p_3\}, \{p_4\})\)
- \((p_2, \{p_3\}, \{p_4\})\)
- \((p_3, \{p_2\}, \emptyset)\)
- \((p_4, \{p_2, p_3\}, \emptyset)\)

MaxSAT formulation:

\[
\varphi_H = \{(\neg x_1 \lor x_2 \lor x_3), (\neg x_1 \lor \neg x_4), (\neg x_2 \lor x_3), (\neg x_2 \lor \neg x_4), (\neg x_3 \lor x_2), (\neg x_4 \lor x_2), (\neg x_4 \lor x_3)\}
\]

\[
\varphi_S = \{(x_1), (x_2), (x_3), (x_4)\}
\]
Minimum Vertex Cover

- The problem:
  - Graph $G = (V, E)$
  - Vertex cover $U \subseteq V$, such that for each edge $(v_i, v_j)$, either $v_i \in U$ or $v_j \in U$.
  - Minimum vertex cover: vertex cover $U$ of minimum size
Minimum Vertex Cover

- The problem:
  - Graph $G = (V, E)$
  - Vertex cover $U \subseteq V$, such that for each edge $(v_i, v_j)$, either $v_i \in U$ or $v_j \in U$.
  - Minimum vertex cover: vertex cover $U$ of minimum size

- Partial MaxSAT formulation:
  - Associate $x_i$ with each $v_i \in V$, such that $x_i = 1$ iff $v_i \in U$, otherwise $v_i \in V - U$.
  - Hard clauses: $(x_i \lor x_j)$ for each edge $(v_i, v_j) \in E$
  - Soft clauses: $(\neg x_i)$ for each vertex $v_i$
    - I.e. preferable not to include vertices in $U$
Minimum Vertex Cover

- The problem:
  - Graph $G = (V, E)$
  - Vertex cover $U \subseteq V$, such that for each edge $(v_i, v_j)$, either $v_i \in U$ or $v_j \in U$.
  - Minimum vertex cover: vertex cover $U$ of minimum size

- Partial MaxSAT formulation:
  - Associate $x_i$ with each $v_i \in V$, such that $x_i = 1$ iff $v_i \in U$, otherwise $v_i \in V - U$.
  - **Hard** clauses: $(x_i \lor x_j)$ for each edge $(v_i, v_j) \in E$
  - **Soft** clauses: $(\neg x_i)$ for each vertex $v_i$
    - i.e. preferable not to include vertices in $U$

\[ \varphi_H = \{(x_1 \lor x_2), (x_1 \lor x_3), (x_1 \lor x_4)\} \]
\[ \varphi_S = \{(-x_1), (-x_2), (-x_3), (-x_4)\} \]
Minimum Vertex Cover

- The problem:
  - Graph $G = (V, E)$
  - Vertex cover $U \subseteq V$, such that for each edge $(v_i, v_j)$, either $v_i \in U$ or $v_j \in U$.
  - Minimum vertex cover: vertex cover $U$ of minimum size

- Partial MaxSAT formulation:
  - Associate $x_i$ with each $v_i \in V$, such that $x_i = 1$ iff $v_i \in U$, otherwise $v_i \in V - U$.
  - Hard clauses: $(x_i \lor x_j)$ for each edge $(v_i, v_j) \in E$
  - Soft clauses: $(\neg x_i)$ for each vertex $v_i$

  - Hard clauses:
    $$\varphi_H = \{(x_1 \lor x_2), (x_1 \lor x_3), (x_1 \lor x_4)\}$$
  - Soft clauses:
    $$\varphi_S = \{(\neg x_1), (\neg x_2), (\neg x_3), (\neg x_4)\}$$
Design Debugging

Correct circuit

Input stimuli: \( \langle r, s \rangle = \langle 0, 1 \rangle \)
Valid output: \( \langle y, z \rangle = \langle 0, 0 \rangle \)

Faulty circuit

Input stimuli: \( \langle r, s \rangle = \langle 0, 1 \rangle \)
Invalid output: \( \langle y, z \rangle = \langle 0, 0 \rangle \)

• The model:
  – Hard clauses: Input and output values
  – Soft clauses: CNF representation of circuit

• The problem:
  – Maximize number of satisfied clauses (i.e. circuit gates)
Binate Covering

- The problem:

\[
\begin{align*}
\min & \quad \sum_{j=1}^{n} c_j x_j \\
\text{s. t.} & \quad \varphi = 1
\end{align*}
\]

- Widely used in Electronic Design Automation (EDA)
  - Logic synthesis (unate); FSM synthesis; etc.

- Weighted partial MaxSAT formulation:
  - Clauses in \( \varphi \) are hard clauses
  - Weighted soft clauses: \((\neg x_j)\) with weight \( c_j \), \( j = 1, \ldots, n \)
    - If negated literals are maximized, then positive literals are minimized
  - If \( c_j = 1 \), then instance of partial MaxSAT
Outline

Boolean-Based Optimization

Practical Applications

Boolean Optimization Algorithms

CNF Encodings
  - Cardinality Constraints
  - Pseudo-Boolean Constraints

Conclusions & Future Work
PBO Algorithms

- **Iterative upper bound refinement** [e.g. Barth’95; Aloul et al.’02; Een&Sorensson’06]
  - Let $B$ be an upper bound on $\sum_{j \in N} c_j \cdot x_j$
  - Create constraint: $\sum_{j \in N} c_j \cdot x_j \leq B - 1$
  - Solve with PB solver
  - If SAT, upper bound updated to $B - 1$
  - Repeat while SAT
PBO Algorithms

- **Iterative upper bound refinement** [e.g. Barth’95; Aloul et al.’02; Een&Sorensson’06]
  - Let $B$ be an upper bound on $\sum_{j \in N} c_j \cdot x_j$
  - Create constraint: $\sum_{j \in N} c_j \cdot x_j \leq B - 1$
  - Solve with PB solver
  - If SAT, upper bound updated to $B - 1$
  - Repeat while SAT

- **Iterative lower bound refinement** [e.g. Fu&Malik’06; Morgado&Marques-Silva’10]
  - Plain or unsatisfiable core guided solutions

- **Binary search on values of cost function** [e.g. Fu&Malik’06]
PBO Algorithms

- **Iterative upper bound refinement** [e.g. Barth’95; Aloul et al.’02; Een&Sorensson’06]
  - Let $B$ be an upper bound on $\sum_{j \in N} c_j \cdot x_j$
  - Create constraint: $\sum_{j \in N} c_j \cdot x_j \leq B - 1$
  - Solve with PB solver
  - If SAT, upper bound updated to $B - 1$
  - Repeat while SAT

- **Iterative lower bound refinement** [e.g. Fu&Malik’06; Morgado&Marques-Silva’10]
  - Plain or unsatisfiable core guided solutions

- **Binary search on values of cost function** [e.g. Fu&Malik’06]

- **Branch-and-bound search** [e.g. Manquinho&Marques-Silva’00]
  - Maintain upper bound ($UB$) of optimum solution
  - Compute estimates of lower bound ($LB$)
  - Bound search if $LB \geq UB$
  - **Must** integrate SAT techniques
MaxSAT Algorithms

- Branch and bound search
  - Sophisticated lower bounds (on \# of unsatisfiable clauses)
    - Unit propagation
    - Inconsistent subsets
  - Dedicated inference techniques
    - Adapted resolution rule
    - ...

- Cannot use most effective SAT techniques
  - No unit propagation
  - No unrestricted clause learning
  - ...

[Li, Manya & Planes’05]
[Li, Manya & Planes’06]
[Heras & Larrosa’06]
MaxSAT Algorithms

• Branch and bound search
  – Sophisticated lower bounds (on # of unsatisfiable clauses)
    ▶ Unit propagation
    ▶ Inconsistent subsets
  – Dedicated inference techniques
    ▶ Adapted resolution rule
    ▶ ...
  – Cannot use most effective SAT techniques
    ▶ No unit propagation
    ▶ No unrestricted clause learning
    ▶ ...

• Translate to PBO

• Core-guided algorithms
Solving MaxSAT with PB I

- Formula $\varphi$ with $n$ variables and $m = |\varphi|$ clauses
- Create $\varphi'$ from $\varphi$:
  - Replace each clause $\omega_i$ with $\omega'_i = \omega_i \cup \{b_i\}$
  - One new relaxation (or blocking) variable $b_i$ for each clause
  - $\varphi'$ with $m$ clauses and $n + m$ variables
- Trivial to satisfy $\varphi'$ by assigning $b_i = 1$, for all $i$
Solving MaxSAT with PB I

- Formula \( \varphi \) with \( n \) variables and \( m = |\varphi| \) clauses
- Create \( \varphi' \) from \( \varphi \):
  - Replace each clause \( \omega_i \) with \( \omega'_i = \omega_i \cup \{b_i\} \)
  - One new relaxation (or blocking) variable \( b_i \) for each clause
  - \( \varphi' \) with \( m \) clauses and \( n + m \) variables
- Trivial to satisfy \( \varphi' \) by assigning \( b_i = 1 \), for all \( i \)

- Example:
  - CNF formula \( \varphi \):
    \[
    \varphi = \{\{x_1, \neg x_2\}, \{x_1, x_2\}, \{\neg x_1\}\}
    \]
  - Modified formula \( \varphi' \):
    \[
    \varphi' = \{\{x_1, \neg x_2, b_1\}, \{x_1, x_2, b_2\}, \{\neg x_1, b_3\}\}
    \]
Iteratively solve the modified PB/SAT problem:

$$\varphi'' = \varphi' \cup \left\{ \sum b_i \leq k \right\}$$

- Start with $k = m - 1$
- Decrease value of $k$ while $\varphi''$ is satisfiable
- Stop when $\varphi''$ becomes unsatisfiable
  - Return $k + 1$ (i.e. value of $k$ for last satisfiable $\varphi''$)
Solving MaxSAT with PB II

• Iteratively solve the modified PB/SAT problem:

\[ \varphi'' = \varphi' \cup \left\{ \sum b_i \leq k \right\} \]

- Start with \( k = m - 1 \)
- Decrease value of \( k \) while \( \varphi'' \) is satisfiable
- Stop when \( \varphi'' \) becomes unsatisfiable
  - Return \( k + 1 \) (i.e. value of \( k \) for last satisfiable \( \varphi'' \))

• Example:
  - Recall modified formula \( \varphi' = \left\{ \left\{ x_1, \neg x_2, b_1 \right\}, \left\{ x_1, x_2, b_2 \right\}, \left\{ \neg x_1, b_3 \right\} \right\} \)
  - Instances of PB/SAT to solve, w/ \( k = 2, 1, 0 \):

\[ \varphi'' = \left\{ \left\{ x_1, \neg x_2, b_1 \right\}, \left\{ x_1, x_2, b_2 \right\}, \left\{ \neg x_1, b_3 \right\} \right\} \cup \left\{ \sum b_i \leq k \right\} \]

  - Satisfiable for \( k = 2, 1 \); Unsatisfiable for \( k = 0 \)
  - Return \( k = 1 \)
Core-Guided MaxSAT – MSU1

Example CNF formula (using ',', instead of '∨')

$x_6, x_2$
$\neg x_6, x_2$
$\neg x_2, x_1$
$\neg x_1$

$\neg x_6, x_8$
$x_6, \neg x_8$
$x_2, x_4$
$\neg x_4, x_5$

$x_7, x_5$
$\neg x_7, x_5$
$\neg x_5, x_3$
$\neg x_3$
Core-Guided MaxSAT – MSU1

Formula is \textbf{UNSAT}; Get unsat core
Core-Guided MaxSAT – MSU1

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_6, x_2$</td>
<td>$\neg x_6, x_2$</td>
<td>$\neg x_2, x_1, b_1$</td>
<td>$\neg x_1, b_2$</td>
<td></td>
</tr>
<tr>
<td>$\neg x_6, x_8$</td>
<td>$x_6, \neg x_8$</td>
<td>$x_2, x_4, b_3$</td>
<td>$\neg x_4, x_5, b_4$</td>
<td></td>
</tr>
<tr>
<td>$x_7, x_5$</td>
<td>$\neg x_7, x_5$</td>
<td>$\neg x_5, x_3, b_5$</td>
<td>$\neg x_3, b_6$</td>
<td></td>
</tr>
</tbody>
</table>

$$\sum_{i=1}^{6} b_i \leq 1$$

Add relaxation variables and AtMost1 constraint
Core-Guided MaxSAT – MSU1

| $x_6, x_2$ | $\neg x_6, x_2$ | $\neg x_2, x_1, b_1$ | $\neg x_1, b_2$ |
| $\neg x_6, x_8$ | $x_6, \neg x_8$ | $x_2, x_4, b_3$ | $\neg x_4, x_5, b_4$ |
| $x_7, x_5$ | $\neg x_7, x_5$ | $\neg x_5, x_3, b_5$ | $\neg x_3, b_6$ |

$$\sum_{i=1}^{6} b_i \leq 1$$

Formula is (again) **UNSAT**; Get unsat core
Core-Guided MaxSAT – MSU1

\[ x_6, x_2, b_7 \quad \neg x_6, x_2, b_8 \quad \neg x_2, x_1, b_1, b_9 \quad \neg x_1, b_2, b_{10} \]

\[ \neg x_6, x_8 \quad x_6, \neg x_8 \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4 \]

\[ x_7, x_5, b_{11} \quad \neg x_7, x_5, b_{12} \quad \neg x_5, x_3, b_5, b_{13} \quad \neg x_3, b_6, b_{14} \]

\[ \sum_{i=1}^{6} b_i \leq 1 \quad \sum_{i=7}^{14} b_i \leq 1 \]

Add new relaxation variables and AtMost1 constraint
Core-Guided MaxSAT – MSU1

\[ x_6, x_2, b_7 \quad \neg x_6, x_2, b_8 \quad \neg x_2, x_1, b_1, b_9 \quad \neg x_1, b_2, b_{10} \]

\[ \neg x_6, x_8 \quad x_6, \neg x_8 \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4 \]

\[ x_7, x_5, b_{11} \quad \neg x_7, x_5, b_{12} \quad \neg x_5, x_3, b_5, b_{13} \quad \neg x_3, b_6, b_{14} \]

\[ \sum_{i=1}^{6} b_i \leq 1 \quad \sum_{i=7}^{14} b_i \leq 1 \]

Instance is now SAT
Core-Guided MaxSAT – MSU1

\[
x_6, x_2, b_7 \quad \neg x_6, x_2, b_8 \quad \neg x_2, x_1, b_1, b_9 \quad \neg x_1, b_2, b_{10}
\]

\[
\neg x_6, x_8 \quad x_6, \neg x_8 \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4
\]

\[
x_7, x_5, b_{11} \quad \neg x_7, x_5, b_{12} \quad \neg x_5, x_3, b_5, b_{13} \quad \neg x_3, b_6, b_{14}
\]

\[
\sum_{i=1}^{6} b_i \leq 1 \quad \sum_{i=7}^{14} b_i \leq 1
\]

MaxSAT solution is \( |\varphi| - I = 12 - 2 = 10 \)
Organization of MSU1

[Fu&Malik'06; Marques-Silva&Manquinho’08; etc.]

- Clauses characterized as:
  - **Soft**: derived from original soft clauses in $\varphi$
  - **Hard**: initial hard or added during execution of algorithm
    - E.g. clauses from cardinality constraints

- While exist unsatisfiable cores
  - Add fresh set $B$ of relaxation variables to soft clauses in core
  - Add new AtMost1 constraint
    \[
    \sum_{b_i \in B} b_i \leq 1
    \]
    - At most 1 relaxation variable from set $B$ can take value 1

- MaxSAT solution is $|\varphi| - I$, where $I$ is number of iterations
  - Minimum number of clauses that must be relaxed for $\varphi$ to be satisfiable
Outline

Boolean-Based Optimization

Practical Applications

Boolean Optimization Algorithms

CNF Encodings
   Cardinality Constraints
   Pseudo-Boolean Constraints

Conclusions & Future Work
Outline

Boolean-Based Optimization

Practical Applications

Boolean Optimization Algorithms

CNF Encodings
   Cardinality Constraints
   Pseudo-Boolean Constraints

Conclusions & Future Work
Cardinality Constraints

• How to handle cardinality constraints, $\sum_{j=1}^{n} x_j \leq k$ ?
  – How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$ ?

• Solution #1:
  – Use PB solver
  – Difficult to keep up with advances in SAT technology
  – For SAT/UNSAT, best PB solvers already encode to SAT
Cardinality Constraints

- How to handle cardinality constraints, $\sum_{j=1}^{n} x_j \leq k$? 
  - How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$?

- Solution #1:
  - Use PB solver
  - Difficult to keep up with advances in SAT technology
  - For SAT/UNSAT, best PB solvers already encode to SAT

- Solution #2:
  - Encode cardinality constraints to CNF
  - Use SAT solver
Equals1, AtLeast1 & AtMost1 Constraints

\[ \sum_{j=1}^{n} x_j = 1: \text{ encode with } (\sum_{j=1}^{n} x_j \leq 1) \land (\sum_{j=1}^{n} x_j \geq 1) \]

\[ \sum_{j=1}^{n} x_j \geq 1: \text{ encode with } (x_1 \lor x_2 \lor \ldots \lor x_n) \]

\[ \sum_{j=1}^{n} x_j \leq 1 \text{ encode with:} \]

- Pairwise encoding
  - Clauses: \( O(n^2) \); No auxiliary variables
- Sequential counter
  - Clauses: \( O(n) \); Auxiliary variables: \( O(n) \)

- Bitwise encoding
  - Clauses: \( O(n \log n) \); Auxiliary variables: \( O(\log n) \)

- ...
Bitwise Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Define $r = \lceil \log n \rceil$ (with $n > 1$); Auxiliary variables $v_0, \ldots, v_{r-1}$
  - Associate with $x_j$ the binary representation of $j-1$
  - Create clauses $(-x_j \lor p_i)$, $i = 0, \ldots, r-1$, where
    - $p_i = v_i$ if the binary representation of $j-1$ has value 1 in position $i$
    - $p_i = \neg v_i$ otherwise
  - If $x_j = 1$, assignment to $v_i$ variables must encode $j-1$
    - All other $x$ variables take value 0
  - If all $x_j = 0$, any assignment to $v_i$ variables is consistent
  - $O(n \log n)$ clauses; $O(\log n)$ auxiliary variables

- An example: $x_1 + x_2 + x_3 \leq 1$

<table>
<thead>
<tr>
<th>$j-1$</th>
<th>$v_1$</th>
<th>$v_0$</th>
<th>Clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0</td>
<td>00</td>
<td>$(-x_1 \lor \neg v_1) \land (-x_1 \lor \neg v_0)$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>1</td>
<td>01</td>
<td>$(-x_2 \lor \neg v_1) \land (-x_2 \lor v_0)$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>2</td>
<td>10</td>
<td>$(-x_3 \lor v_1) \land (-x_3 \lor \neg v_0)$</td>
</tr>
</tbody>
</table>
General Cardinality Constraints

- General form: $\sum_{j=1}^{n} x_j \leq k$
  - Sequential counter
    - Clauses/Variables: $O(nk)$
  - BDDs
    - Clauses/Variables: $O(nk)$
  - Sorting networks
    - Clauses/Variables: $O(n \log^2 n)$
  - Cardinality Networks:
    - Clauses/Variables: $O(n \log^2 k)$
  - ...

[Sinz'05]
[Een&Sorensson'06]
[Een&Sorensson'06]
[Asin et al.'09]
• Encode $\sum_{j=1}^{n} x_j \leq k$ with sorting network:
  – Unary representation
  – Use **odd-even merging networks** [Batcher’68; Een&Sorensson’06; Asin et al.’09; etc.]
  – Recursive definition of **merging networks**
Encode $\sum_{j=1}^{n} x_j \leq k$ with sorting network:
- Unary representation
- Use odd-even merging networks [Batcher’68; Een&Sorensson’06; Asin et al.’09; etc.]
- Recursive definition of merging networks
  - Base Case:
    $$\text{Merge}(a_1, b_1) \triangleq (\langle c_1, c_2 \rangle, \{c_2 = \min(a_1, b_1), c_1 = \max(a_1, b_1)\})$$
• Encode $\sum_{j=1}^{n} x_j \leq k$ with sorting network:
  - Unary representation
  - Use **odd-even merging networks** [Batcher’68; Een&Sorensson’06; Asin et al.’09; etc.]
  - Recursive definition of **merging networks**
    - Base Case:
      $\text{Merge}(a_1, b_1) \triangleq (\langle c_1, c_2 \rangle, \{c_2 = \min(a_1, b_1), c_1 = \max(a_1, b_1)\})$
    - Let:
      $\text{Merge}(\langle a_1, a_3, \ldots, a_{n-1} \rangle, \langle b_1, b_3, \ldots, b_{n-1} \rangle) \triangleq (\langle d_1, \ldots, d_n \rangle, S_{\text{odd}})$
      $\text{Merge}(\langle a_2, a_4, \ldots, a_n \rangle, \langle b_2, b_4, \ldots, b_n \rangle) \triangleq (\langle e_1, \ldots, e_n \rangle, S_{\text{even}})$
• Encode $\sum_{j=1}^{n} x_j \leq k$ with sorting network:
  - Unary representation
  - Use odd-even merging networks [Batcher’68; Een&Sorensson’06; Asin et al.’09; etc.]
  - Recursive definition of merging networks
    ▶ Base Case:
    $\text{Merge}(a_1, b_1) \triangleq (\langle c_1, c_2 \rangle, \{ c_2 = \min(a_1, b_1), c_1 = \max(a_1, b_1) \})$
    ▶ Let:
    $\text{Merge}(\langle a_1, a_3, \ldots, a_{n-1} \rangle, \langle b_1, b_3, \ldots, b_{n-1} \rangle) \triangleq (\langle d_1, \ldots, d_n \rangle, S_{\text{odd}})$
    $\text{Merge}(\langle a_2, a_4, \ldots, a_n \rangle, \langle b_2, b_4, \ldots, b_n \rangle) \triangleq (\langle e_1, \ldots, e_n \rangle, S_{\text{even}})$
    ▶ Then:
    $\text{Merge}(\langle a_1, a_2, \ldots, a_n \rangle, \langle b_1, b_2, \ldots, b_n \rangle) \triangleq$
    $(\langle d_1, c_1, \ldots, c_{2n-1}, e_n \rangle, S_{\text{odd}} \cup S_{\text{even}} \cup S_{\text{mrg}})$
    ▶ Where:
    $S_{\text{mrg}} = \bigcup_{i=1}^{n-1} \{ c_{2i+1} = \min(d_{i+1}, e_i), c_{2i} = \max(d_{i+1}, e_i) \}$
• Recursive definition of sorting networks
  – Base case ($n = 2$):
    \[
    \text{Sort}(a_1, a_2) \triangleq \text{Merge}(a_1, a_2)
    \]
Recursive definition of sorting networks

- Base case ($n = 2$):
  \[ \text{Sort}(a_1, a_2) \triangleq \text{Merge}(a_1, a_2) \]

- Let:
  \[ \text{Sort}(\langle a_1, \ldots, a_n \rangle) \triangleq (\langle d_1, \ldots, d_n \rangle, S_D) \]
  \[ \text{Sort}(\langle a_{n+1}, \ldots, a_{2n} \rangle) \triangleq (\langle d'_1, \ldots, d'_n \rangle, S'_D) \]
  and,
  \[ \text{Merge}(\langle d_1, \ldots, d_n \rangle, \langle d'_1, \ldots, d'_n \rangle) \triangleq (\langle c_1, \ldots, c_{2n} \rangle, S_M) \]
• Recursive definition of sorting networks
  
  - Base case ($n = 2$):
    \[ \text{Sort}(a_1, a_2) \triangleq \text{Merge}(a_1, a_2) \]
  - Let:
    \[ \text{Sort}(\langle a_1, \ldots, a_n \rangle) \triangleq (\langle d_1, \ldots, d_n \rangle, S_D) \]
    \[ \text{Sort}(\langle a_{n+1}, \ldots, a_{2n} \rangle) \triangleq (\langle d'_1, \ldots, d'_n \rangle, S'_D) \]
    and,
    \[ \text{Merge}(\langle d_1, \ldots, d_n \rangle, \langle d'_1, \ldots, d'_n \rangle) \triangleq (\langle c_1, \ldots, c_{2n} \rangle, S_M) \]
  - Then:
    \[ \text{Sort}(a_1, \ldots, a_{2n}) \triangleq (\langle c_1, \ldots, c_{2n} \rangle, S_D \cup S'_D \cup S_M) \]
Recursive definition of sorting networks

- Base case \((n = 2)\):
  \[
  \text{Sort}(a_1, a_2) \triangleq \text{Merge}(a_1, a_2)
  \]
- Let:
  \[
  \text{Sort}\left(\langle a_1, \ldots, a_n \rangle\right) \triangleq \left(\langle d_1, \ldots, d_n \rangle, S_D\right)
  \]
  \[
  \text{Sort}\left(\langle a_{n+1}, \ldots, a_{2n} \rangle\right) \triangleq \left(\langle d'_1, \ldots, d'_n \rangle, S'_D\right)
  \]
  and,
  \[
  \text{Merge}\left(\langle d_1, \ldots, d_n \rangle, \langle d'_1, \ldots, d'_n \rangle\right) \triangleq \left(\langle c_1, \ldots, c_{2n} \rangle, S_M\right)
  \]
- Then:
  \[
  \text{Sort}(a_1, \ldots, a_{2n}) \triangleq \left(\langle c_1, \ldots, c_{2n} \rangle, S_D \cup S'_D \cup S_M\right)
  \]

Let \(\langle z_1, \ldots, z_n \rangle\) be the sorted output.
The constraint \((\text{for } \leq k)\) is:
\[z_i = 0, \quad i > k\]
Example: Sort $\langle a_1, a_2, a_3, a_4 \rangle$

where each Merge block contains 1 min ($\equiv$ AND) and 1 max ($\equiv$ OR) operators
Outline

Boolean-Based Optimization

Practical Applications

Boolean Optimization Algorithms

CNF Encodings
  - Cardinality Constraints
  - Pseudo-Boolean Constraints

Conclusions & Future Work
Pseudo-Boolean Constraints

- General form: $\sum_{j=1}^{n} a_j \cdot x_j \leq b$
  - Operational encoding
    - Clauses/Variables: $O(n)$
    - Does not guarantee arc-consistency
  - BDDs
    - Worst-case exponential number of clauses
  - Polynomial watchdog encoding
    - Let $\nu(n) = \log(n) \log(a_{\text{max}})$
    - Clauses: $O(n^3 \nu(n))$; Aux variables: $O(n^2 \nu(n))$
  - BDD-based encoding
    - Clauses: $O(n^3 \log(a_{\text{max}}))$; Aux variables: $O(n^3 \log(a_{\text{max}}))$
- ...
Encoding PB Constraints with BDDs I

- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. by analyzing variables by decreasing coefficients
- Extract ITE-based circuit from BDD
Encoding PB Constraints with BDDs I

- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. by analyzing variables by decreasing coefficients
- Extract ITE-based circuit from BDD
Encoding PB Constraints with BDDs II

- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Extract ITE-based circuit from BDD
- Simplify and create final circuit:
Conclusions

- Well-known **Boolean-based decision procedures**
  - Z3, Yices, BarcelogicTools, Minisat, Picosat, etc.
  - Many practical applications
Conclusions

• Well-known Boolean-based decision procedures
  – Z3, Yices, BarcelogicTools, Minisat, Picosat, etc.
  – Many practical applications

• Upcoming Boolean-based optimization procedures
  – SAT-based: MaxSAT; PBO; WBO
  – SMT-based: MaxSMT
  – ...

Conclusions

• Well-known **Boolean-based decision procedures**
  – Z3, Yices, BarcelogicTools, Minisat, Picosat, etc.
  – Many practical applications

• Upcoming **Boolean-based optimization procedures**
  – SAT-based: MaxSAT; PBO; WBO
  – SMT-based: MaxSMT
  – ...

• Several (**many?**) exciting applications
  – Software package upgrades
  – Bug localization in C code
  – Design debugging
  – ...

[Argelich et al.’09]
[Jose & Majumdar’11]
[Safarpour et al.’07]
Conclusions

- **Well-known Boolean-based decision procedures**
  - Z3, Yices, BarcelogicTools, Minisat, Picosat, etc.
  - Many practical applications

- **Upcoming Boolean-based optimization procedures**
  - SAT-based: MaxSAT; PBO; WBO
  - SMT-based: MaxSMT
  - ...

- **Several (many?) exciting applications**
  - Software package upgrades
  - Bug localization in C code
  - Design debugging
  - ...

- **Many new lines of research**
  - Core-guided approaches
  - CNF encodings of constraints
  - Integration with relaxation-based approaches
  - ...

[Argelich et al.'09]
[Jose & Majumdar'11]
[Safarpour et al.'07]