MaxSAT and Related Optimization Problems

Joao Marques-Silva$^{1,2}$

$^1$University College Dublin, Ireland
$^2$IST/INESC-ID, Lisbon, Portugal

SAT/SMT Summer School 2012
FBK, Trento, Italy
The problem:
- Graph $G = (V, E)$
- Vertex cover $U \subseteq V$
  - For each $(v_i, v_j) \in E$, either $v_i \in U$ or $v_j \in U$
- Minimum vertex cover: vertex cover $U$ of minimum size
Example Problem: Minimum Vertex Cover

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

Vertex cover: $\{v_2, v_3, v_4\}$
Example Problem: Minimum Vertex Cover

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

Vertex cover: $\{v_2, v_3, v_4\}$

Min vertex cover: $\{v_1\}$
Example Problem: Minimum Vertex Cover

- **Pseudo-Boolean Optimization (PBO) formulation:**
  - Variables: \( x_i \) for each \( v_i \in V \), with \( x_i = 1 \) iff \( v_i \in U \)
  - Clauses: \((x_i \vee x_j)\) for each \((v_i, v_j) \in E\)
  - Objective function: minimize number of true \( x_i \) variables
    - i.e. minimize vertices included in \( U \)
Example Problem: Minimum Vertex Cover

- **Pseudo-Boolean Optimization (PBO) formulation:**
  - **Variables:** $x_i$ for each $v_i \in V$, with $x_i = 1$ iff $v_i \in U$
  - **Clauses:** $(x_i \lor x_j)$ for each $(v_i, v_j) \in E$
  - **Objective function:** minimize number of true $x_i$ variables
    - i.e. minimize vertices included in $U$

\[
\begin{align*}
\text{minimize} & \quad x_1 + x_2 + x_3 + x_4 \\
\text{subject to} & \quad (x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_4)
\end{align*}
\]
Boolean-Based Optimization

- Linear optimization over Boolean domains
Boolean-Based Optimization

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

- **Linear optimization over Boolean domains**
  - **Note:** 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

[eg. HLO’08]
Boolean-Based Optimization

- **Linear optimization over Boolean domains**
  - **Note:** 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 [e.g. HLO’08]

- **Related problems:**
  - Optimization in SMT (MaxSMT)
  - Optimization in CSP (MaxCSP, etc.)
  - Integer Linear Programming (ILP)
Different ways of representing Boolean optimization problems are (essentially) equivalent
- Pseudo-Boolean Optimization (PBO) (or 0-1 ILP)
- Maximum Satisfiability (MaxSAT)
- Weighted Boolean Optimization (WBO)
- etc.

Optimization algorithms can (and do!) build on SAT solver technology
- By using PBO
- By using Core-guided MaxSAT

Algorithms for MaxSAT can be readily extended to MaxSMT
Outline

Boolean-Based Optimization

Example Applications

Fundamental Techniques

Practical Algorithms

Results, Conclusions & Research Directions
Boolean-Based Optimization

Example Applications

Fundamental Techniques

Practical Algorithms

Results, Conclusions & Research Directions
What is Maximum Satisfiability?

- CNF Formula:

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

- CNF Formula:

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

- Formula is unsatisfiable
- MaxSAT:
  - Find assignment that maximizes number of satisfied clauses
    - For above formula, solution is 10
- There are a number of variants of MaxSAT
MaxSAT Problem(s)

- **MaxSAT:**
  - All clauses are *soft*
  - Maximize number of *satisfied soft* clauses
  - Minimize number of *unsatisfied soft* clauses
MaxSAT Problem(s)

• MaxSAT:
  – All clauses are soft
  – Maximize number of satisfied soft clauses
  – Minimize number of unsatisfied soft clauses

• Partial MaxSAT:
  – Hard clauses must be satisfied
  – Minimize number of unsatisfied soft clauses
MaxSAT Problem(s)

- **MaxSAT:**
  - All clauses are soft
  - Maximize number of satisfied soft clauses
  - Minimize number of unsatisfied soft clauses

- **Partial MaxSAT:**
  - Hard clauses must be satisfied
  - Minimize number of unsatisfied soft clauses

- **Weighted MaxSAT**
  - Weights associated with (soft) clauses
  - Minimize sum of weights of unsatisfied clauses
MaxSAT Problem(s)

- **MaxSAT**:  
  - All clauses are soft  
  - Maximize number of satisfied soft clauses  
  - Minimize number of unsatisfied soft clauses

- **Partial MaxSAT**:  
  - Hard clauses must be satisfied  
  - Minimize number of unsatisfied soft clauses

- **Weighted MaxSAT**:  
  - Weights associated with (soft) clauses  
  - Minimize sum of weights of unsatisfied clauses

- **Weighted Partial MaxSAT**:  
  - Weights associated with soft clauses  
  - Hard clauses must be satisfied  
  - Minimize sum of weights of unsatisfied soft clauses
Complexity of MaxSAT

- (decision version of) MaxSAT is \textit{NP-complete}
- Solving MaxSAT with calls to a \textit{SAT oracle}
Complexity of MaxSAT

- (decision version of) MaxSAT is NP-complete

- Solving MaxSAT with calls to a SAT oracle
  - (Unweighted) MaxSAT is $\Delta^p_2[\log n]$-complete
    - Logarithmic number of calls (on instance size) for unweighted MaxSAT
  - Weighted MaxSAT is $\Delta^p_2$-complete
    - Linear number of calls (on instance size) for weighted MaxSAT
MaxSAT Notation

- \((c_i, w_i)\): weighted clause
  - \(c_i\) is a set of literals (clause)
  - \(w_i\) is a non-negative integer or \(\infty\) (or \(\top\))
    - Cost of not satisfying \(c_i\)

- \(\varphi\): set of weighted clauses
  - Soft clauses: \((c_i, w_i)\), with \(w_i < \infty\)
    - Cost of not satisfying \(c_i\) is \(w_i\)
  - Hard clauses: \((c_i, \infty)\)
    - Clause \(c_i\) must be satisfied
Modeling Example: Minimum Vertex Cover

• Partial MaxSAT formulation:
  - Variables: $x_i$ for each $v_i \in V$, with $x_i = 1$ iff $v_i \in U$
  - Hard clauses: $(x_i \lor x_j)$ for each $(v_i, v_j) \in E$
  - Soft clauses: $(\neg x_i)$ for each $v_i \in V$
    ▶ I.e. prefer not to include vertices in $U$
Partial MaxSAT formulation:
- Variables: $x_i$ for each $v_i \in V$, with $x_i = 1$ iff $v_i \in U$
- **Hard** clauses: $(x_i \lor x_j)$ for each $(v_i, v_j) \in E$
- **Soft** clauses: $(\neg x_i)$ for each $v_i \in V$
  - I.e. prefer not to include vertices in $U$

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

- **Hard** clauses have cost $\infty$
- **Soft** clauses have cost 1
Pseudo-Boolean Constraints & Optimization

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

- **Pseudo-Boolean Optimization (PBO):**
  
  minimize \[ \sum_{j \in N} w_j \cdot x_j \]

  subject to \[ \sum_{j \in N} a_{ij} l_j \geq b_i, \]

  \[ l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0, 1\}, a_{ij}, b_i, w_j \in \mathbb{N}_0^+ \]
Solving MaxSAT with PBO – Unweighted

• Create \( \varphi' \) from \( \varphi \):
  – Replace each \( c_i \) with \( c'_i = c_i \cup \{ r_i \} \)
    - Fresh relaxation variable \( r_i \) for each clause \( c_i \)
  – Note: Trivial to satisfy \( \varphi' \) by assigning \( r_i = 1 \), for all \( i \)

• Minimize cost function: \( \sum r_i \)
Solving MaxSAT with PBO – Unweighted

- Create \( \varphi' \) from \( \varphi \):
  - Replace each \( c_i \) with \( c'_i = c_i \cup \{r_i\} \)
    - Fresh relaxation variable \( r_i \) for each clause \( c_i \)
  - Note: Trivial to satisfy \( \varphi' \) by assigning \( r_i = 1 \), for all \( i \)

- Minimize cost function: \( \sum r_i \)

- Example:
  - CNF formula \( \varphi \):
    \[
    \varphi = \{\{x_1, \neg x_2\}, \{x_1, x_2\}, \{\neg x_1\}\}
    \]
  - Modified formula \( \varphi' \):
    \[
    \varphi' = \{\{x_1, \neg x_2, r_1\}, \{x_1, x_2, r_2\}, \{\neg x_1, r_3\}\}
    \]
  - Minimize cost function: \( r_1 + r_2 + r_3 \)
Solving MaxSAT with PBO – General Case

- MaxSAT instance:
  - \( \varphi_H \): hard clauses of the form \((c_i, \infty)\)
  - \( \varphi_S \): (weighted) soft clauses of the form \((c_i, w_i)\)
Solving MaxSAT with PBO – General Case

- **MaxSAT instance:**
  - $\varphi_H$: hard clauses of the form $(c_i, \infty)$
  - $\varphi_S$: (weighted) soft clauses of the form $(c_i, w_i)$

- **Create PBO instance:**

\[
\min \sum w_i r_i \\
\text{s.t.} \quad \varphi_T
\]

where,

- $\varphi_T = \varphi'_H \cup \varphi'_S$
- $\varphi'_H$: Each **hard** clause $(c_i, \infty) \in \varphi_H$ is mapped into clause $c_i$ in $\varphi_T$
- $\varphi'_S$: Each **soft** clause $(c_i, w_i)$ is mapped into a clause $(c_i \lor r_i)$, and term $w_i r_i$ is added to cost function
Solving PBO with MaxSAT – General Case

- **Binate covering instance:**

\[
\begin{align*}
\text{min} & \quad \sum w_i x_i \\
\text{s.t.} & \quad \varphi
\end{align*}
\]
Solving PBO with MaxSAT – General Case

• Binate covering instance:

\[
\min \sum w_i x_i \\
\text{s.t. } \varphi
\]

• Create MaxSAT instance:
  - \( \varphi_H \triangleq \varphi \): hard clauses of the form \((c_i, \infty)\)
  - \( \varphi_S \): for each cost function term \(w_i x_i\), create soft clause \((\neg x_i, w_i)\)

• General PB constraints?
  - Encode PB constraints to CNF, or
  - Use Weighted Boolean Optimization [MMSP'09]
Outline

Boolean-Based Optimization

Example Applications

Fundamental Techniques

Practical Algorithms

Results, Conclusions & Research Directions
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)
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:

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

$\varphi_S = \{(x_1), (x_2), (x_3), (x_4)\}$
Key Engine for MUS Enumeration

- **MUS**: irreducible unsatisfiable set of clauses
  - **MCS**: irreducible set of clauses such that complement is satisfiable
  - **MSS**: subset maximal satisfiable set of clauses
Key Engine for MUS Enumeration

- **MUS**: irreducible unsatisfiable set of clauses
  - **MCS**: irreducible set of clauses such that complement is satisfiable
  - **MSS**: subset maximal satisfiable set of clauses

- Enumeration of MUSes finds many applications:
  - Model checking with CEGAR, type inference & checking, etc. [ALS'08,BSW'03]
Key Engine for MUS Enumeration

- **MUS**: irreducible unsatisfiable set of clauses
  - **MCS**: irreducible set of clauses such that complement is satisfiable
  - **MSS**: subset maximal satisfiable set of clauses

- Enumeration of MUSes finds **many** applications:
  - Model checking with CEGAR, type inference & checking, etc.  
    \[\text{[ALS'08,BSW'03]}\]

- **How to enumerate MUSes?**
  - Use **hitting set duality** between MUSes and MCSes
    - An MUS is an irreducible hitting set of a formula’s MCSes
    - An MCS is an irreducible hitting set of a formula’s MUSes
  - Can enumerate MCSes and then use them to compute MUSes
    \[\text{[E.g. LS'08]}\]  
    \[\text{[E.g. R'87,BL'03]}\]
Key Engine for MUS Enumeration

- **MUS**: irreducible unsatisfiable set of clauses
  - **MCS**: irreducible set of clauses such that complement is satisfiable
  - **MSS**: subset maximal satisfiable set of clauses

- Enumeration of MUSes finds many applications:
  - Model checking with CEGAR, type inference & checking, etc. [ALS’08, BSW’03]

- How to enumerate MUSes?
  - Use hitting set duality between MUSes and MCSes [E.g. R’87, BL’03]
    - An MUS is an irreducible hitting set of a formula’s MCSes
    - An MCS is an irreducible hitting set of a formula’s MUSes
  - Can enumerate MCSes and then use them to compute MUSes
  - Use MaxSAT enumeration for computing all MSSes
Many Other Applications

- Error localization in C code [JM’11]
- Haplotyping with pedigrees [GLMSO’10]
- Course timetabling [AN’10]
- Combinatorial auctions [HLGS’08]
- Minimizing Disclosure of Private Information in Credential-Based Interactions [AVFPS’10]
- Reasoning over Biological Networks [GL’12]
- Binate/unate covering
  - Haplotype inference [GMSLO’11]
  - Digital filter design [ACFM’08]
  - FSM synthesis [e.g. HS’96]
  - Logic minimization [e.g. HS’96]
  - ...
- ...
- ...
- ...
Outline

Boolean-Based Optimization

Example Applications

Fundamental Techniques

Practical Algorithms

Results, Conclusions & Research Directions
Main Techniques

- **Unit propagation**
  - For computing lower bounds in B&B MaxSAT

- **Stochastic Local Search**
  - For computing upper bounds (e.g. B&B MaxSAT)

- **Unsatisfiable subformulas (or cores)**
  - Used in core-guided MaxSAT algorithms

- **CNF encodings**
  - Cardinality constraints
  - PB constraints
Outline

Boolean-Based Optimization

Example Applications

Fundamental Techniques

Cardinality Constraints

Pseudo-Boolean Constraints

Practical Algorithms

Results, Conclusions & Research Directions
Cardinality Constraints

- How to handle cardinality constraints, $\sum_{j=1}^{n} x_j \leq k$?
  - How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$?
  - General form: $\sum_{j=1}^{n} x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$

- Solution #1:
  - Use PB solver
  - Difficult to keep up with advances in SAT technology
  - For SAT/UNSAT, best solvers already encode to CNF
    - E.g. Minisat+, but also QMaxSat, MSUnCore, (W)PM2
Cardinality Constraints

• How to handle cardinality constraints, $\sum_{j=1}^{n} x_j \leq k$?
  – How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$?
  – General form: $\sum_{j=1}^{n} x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$

• Solution #1:
  – Use PB solver
  – Difficult to keep up with advances in SAT technology
  – For SAT/UNSAT, best solvers already encode to CNF
    ▶ E.g. Minisat+, but also QMaxSat, MSUnCore, (W)PM2

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

- $\sum_{j=1}^{n} x_j = 1$: encode with $(\sum_{j=1}^{n} x_j \leq 1) \land (\sum_{j=1}^{n} x_j \geq 1)$

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

- $\sum_{j=1}^{n} x_j \leq 1$ encode with:
  - Pairwise encoding
    - Clauses: $O(n^2)$; No auxiliary variables
  - Sequential counter [S’05]
    - Clauses: $O(n)$; Auxiliary variables: $O(n)$
  - Bitwise encoding [P’07,FP’01]
    - Clauses: $O(n \log n)$; Auxiliary variables: $O(\log n)$
  - ...
Bitwise Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:

An example: $x_1 + x_2 + x_3 \leq 1$
Bitwise Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{j-1} = b_0 \ldots b_{j-1}$, the binary encoding $j - 1$
    $$x_j \rightarrow (v_0 = b_0) \land \ldots \land (v_{j-1} = b_{j-1}) \iff (\neg x_j \lor (v_0 = b_0) \land \ldots \land (v_{j-1} = b_{j-1}))$$

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

<table>
<thead>
<tr>
<th></th>
<th>$j - 1$</th>
<th>$v_1 \cdot v_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>$x_2$</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>$x_3$</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>
Bitwise Encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{j-1} = b_0 \ldots b_{j-1}$, the binary encoding $j-1$
    \[
    x_j \rightarrow (v_0 = b_0) \land \ldots \land (v_{j-1} = b_{j-1}) \iff \neg x_j \lor (v_0 = b_0) \land \ldots \land (v_{j-1} = b_{j-1})
    \]
  - Clauses $(\neg x_j \lor (v_i \leftrightarrow b_i)) = (\neg x_j \lor l_i)$, $i = 0, \ldots, r-1$, where
    \begin{itemize}
    \item $l_i \equiv v_i$, if $b_i = 1$
    \item $l_i \equiv \neg v_i$, otherwise
    \end{itemize}

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

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

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{j-1} = b_0 \ldots b_{j-1}$, the binary encoding $j-1$
    
    $x_j \rightarrow (v_0 = b_0) \land \ldots \land (v_{j-1} = b_{j-1}) \iff (\neg x_j \lor (v_0 = b_0) \land \ldots \land (v_{j-1} = b_{j-1}))$

  - Clauses $(\neg x_j \lor (v_i \leftrightarrow b_i)) = (\neg x_j \lor l_i)$, $i = 0, \ldots, r - 1$, where
    - $l_i \equiv v_i$, if $b_i = 1$
    - $l_i \equiv \neg v_i$, otherwise
  - If $x_j = 1$, assignment to $v_i$ variables must encode $j - 1$
    - All other $x$ variables must 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>$v_1 v_0$</th>
<th>$(\neg x_1 \lor \neg v_1) \land (\neg x_1 \lor \neg v_0)$</th>
<th>$(\neg x_2 \lor \neg v_1) \land (\neg x_2 \lor v_0)$</th>
<th>$(\neg x_3 \lor v_1) \land (\neg x_3 \lor \neg v_0)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0</td>
<td>00</td>
<td>$(\neg x_1 \lor \neg v_1) \land (\neg x_1 \lor \neg v_0)$</td>
<td></td>
</tr>
<tr>
<td>$x_2$</td>
<td>1</td>
<td>01</td>
<td>$(\neg x_2 \lor \neg v_1) \land (\neg x_2 \lor v_0)$</td>
<td></td>
</tr>
<tr>
<td>$x_3$</td>
<td>2</td>
<td>10</td>
<td>$(\neg x_3 \lor v_1) \land (\neg x_3 \lor \neg v_0)$</td>
<td></td>
</tr>
</tbody>
</table>
General Cardinality Constraints

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

• Encode $\sum_{j=1}^{n} x_j \leq k$ with sequential counter:

$\begin{align*}
\text{s}_i &= \sum_{j=1}^{i} x_j \\
\text{s}_i \text{ represented in unary}
\end{align*}$

• Equations for each block $1 < i < n$, $1 < j < k$:

$\begin{align*}
\text{s}_i &= \sum_{j=1}^{i} x_j \\
\text{s}_{i,1} &= \text{s}_{i-1,1} \lor x_i \\
\text{s}_{i,j} &= \text{s}_{i-1,j} \lor \text{s}_{i-1,j-1} \land x_i \\
\text{v}_i &= (\text{s}_{i-1,k} \land x_i) = 0
\end{align*}$
Sequential Counter

• CNF formula for $\sum_{j=1}^{n} x_j \leq k$:
  
  – Assume: $k > 0 \land n > 1$
  
  – Indices: $1 < i < n$, $1 < j \leq k$

\[
\begin{align*}
(\neg x_1 \lor \neg x_{1,1}) \\
(\neg s_{1,j}) \\
(\neg x_i \lor \neg s_{i,1}) \\
(\neg s_{i-1,1} \lor \neg s_{i,1}) \\
(\neg x_i \lor \neg s_{i-1,j-1} \lor s_{i,j}) \\
(\neg s_{i-1,j} \lor s_{i,j}) \\
(\neg x_i \lor \neg s_{i-1,k}) \\
(\neg x_n \lor \neg s_{n-1,k})
\end{align*}
\]

• $O(nk)$ clauses & variables
• Encode $\sum_{j=1}^{n} x_j \leq k$ with sorting network:
  – Unary representation
  – Use odd-even merging networks
  – Recursive definition of merging networks

[B’68,ES’06,ANORC’09]
• Encode $\sum_{j=1}^{n} x_j \leq k$ with sorting network:
  - Unary representation
  - Use odd-even merging networks
  - 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**
  - 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:**
    
    \[
    \text{Merge}(\langle a_1, a_3, \ldots, a_{n-1} \rangle, \langle b_1, b_3, \ldots, b_{n-1} \rangle) \triangleq (\langle d_1, \ldots, d_n \rangle, S_{odd}) \\
    \text{Merge}(\langle a_2, a_4, \ldots, a_n \rangle, \langle b_2, b_4, \ldots, b_n \rangle) \triangleq (\langle e_1, \ldots, e_n \rangle, S_{even})
    \]
• Encode $\sum_{j=1}^{n} x_j \leq k$ with sorting network:
  - Unary representation
  - Use odd-even merging networks
  - 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:
  \[
  S_{\text{mrg}} = \bigcup_{i=1}^{n-1} \{ c_{2i+1} = \min(d_{i+1}, e_i), c_{2i} = \max(d_{i+1}, e_i) \}
  \]
• Encode $\sum_{j=1}^{n} x_j \leq k$ with sorting network:
  
  - Unary representation
  - Use odd-even merging networks
  - 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:
  
  \[
  \begin{align*}
  \text{Merge}(\langle a_1, a_3, \ldots, a_{n-1} \rangle, \langle b_1, b_3, \ldots, b_{n-1} \rangle) & \triangleq (\langle d_1, \ldots, d_n \rangle, S_{\text{odd}}) \\
  \text{Merge}(\langle a_2, a_4, \ldots, a_n \rangle, \langle b_2, b_4, \ldots, b_n \rangle) & \triangleq (\langle e_1, \ldots, e_n \rangle, S_{\text{even}})
  \end{align*}
  \]

  ▶ 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:
  
  \[
  S_{\text{mrg}} = \bigcup_{i=1}^{n-1} \{c_{2i+1} = \min(d_{i+1}, e_i), c_{2i} = \max(d_{i+1}, e_i)\}
  \]

  - Note: $\min \equiv \text{AND}$ and $\max \equiv \text{OR}$
Recursive definition of sorting networks

- Base Case ($2n = 2$):
  \[ \text{Sort}(a_1, b_1) \triangleq \text{Merge}(a_1, b_1) \]
• Recursive definition of sorting networks
  
  – Base Case ($2n = 2$):
    \[ \text{Sort}(a_1, b_1) \triangleq \text{Merge}(a_1, b_1) \]

  – Inductive Step ($2n > 2$):
    
    Let,
    \[
    \begin{align*}
    \text{Sort}(a_1, \ldots, a_n) & \triangleq (\langle d_1, \ldots, d_n \rangle, S_D) \\
    \text{Sort}(a_{n+1}, \ldots, a_{2n}) & \triangleq (\langle d'_1, \ldots, d'_n \rangle, S'_D) \\
    \text{Merge}(\langle d_1, \ldots, d_n \rangle, \langle d'_1, \ldots, d'_n \rangle) & \triangleq (\langle c_1, \ldots, c_{2n} \rangle, S_M)
    \end{align*}
    \]
Recursive definition of sorting networks

- Base Case ($2n = 2$):

\[ \text{Sort}(a_1, b_1) \triangleq \text{Merge}(a_1, b_1) \]

- Inductive Step ($2n > 2$):

  Let,

\[
\begin{align*}
\text{Sort}(a_1, \ldots, a_n) & \triangleq (\langle d_1, \ldots, d_n \rangle, S_D) \\
\text{Sort}(a_{n+1}, \ldots, a_{2n}) & \triangleq (\langle d'_1, \ldots, d'_n \rangle, S'_D) \\
\text{Merge}(\langle d_1, \ldots, d_n \rangle, \langle d'_1, \ldots, d'_n \rangle) & \triangleq (\langle c_1, \ldots, c_{2n} \rangle, S_M)
\end{align*}
\]

  Then,

\[ \text{Sort}(\langle a_1, \ldots, a_{2n} \rangle) \triangleq (\langle c_1, \ldots, c_{2n} \rangle, S_D \cup S'_D \cup S_M) \]
Recursive definition of sorting networks

- Base Case \((2n = 2)\):
  \[
  \text{Sort}(a_1, b_1) \triangleq \text{Merge}(a_1, b_1)
  \]

- Inductive Step \((2n > 2)\):
  
  - Let,
    \[
    \begin{align*}
    \text{Sort}(a_1, \ldots, a_n) & \triangleq (\langle d_1, \ldots, d_n \rangle, S_D) \\
    \text{Sort}(a_{n+1}, \ldots, a_{2n}) & \triangleq (\langle d'_1, \ldots, d'_n \rangle, S'_D) \\
    \text{Merge}(\langle d_1, \ldots, d_n \rangle, \langle d'_1, \ldots, d'_n \rangle) & \triangleq (\langle c_1, \ldots, c_{2n} \rangle, S_M)
    \end{align*}
    \]
  
  - Then,
    \[
    \text{Sort}(\langle a_1, \ldots, a_{2n} \rangle) \triangleq (\langle c_1, \ldots, c_{2n} \rangle, S_D \cup S'_D \cup S_M)
    \]

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

where each \texttt{Merge} block contains 1 min (AND) and 1 max (OR) operators.
Outline

Boolean-Based Optimization

Example Applications

**Fundamental Techniques**

Cardinality Constraints

Pseudo-Boolean Constraints

Practical Algorithms

Results, Conclusions & Research Directions
Pseudo-Boolean Constraints

- General form: $\sum_{j=1}^{n} a_j x_j \leq b$
  - 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: $O(n^2 \nu(n))$
  - Improved polynomial watchdog encoding
    - Clauses & 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. analyze 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. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD
• Encode $3x_1 + 3x_2 + x_3 \leq 3$

• Extract ITE-based circuit from BDD

• Simplify and create final circuit:
More on PB Constraints

- How about \( \sum_{j=1}^{n} a_j x_j = k \)?
More on PB Constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j x_j = k$ is a subset-sum constraint
      (special case of a knapsack constraint)
How about $\sum_{j=1}^{n} a_j x_j = k$?

- Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
  - $\sum_{j=1}^{n} a_j x_j = k$ is a subset-sum constraint
    (special case of a knapsack constraint)
  - Cannot find all consequences in polynomial time

[S'03,FS'02,T'03]
More on PB Constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j x_j = k$ is a subset-sum constraint
      (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time

- Example:

  $4x_1 + 3x_2 + 2x_3 = 5$
More on PB Constraints

• How about \( \sum_{j=1}^{n} a_j x_j = k \) ?
  
  – Can use \((\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)\), but...
    
    ▶ \( \sum_{j=1}^{n} a_j x_j = k \) is a subset-sum constraint
    (special case of a knapsack constraint)
    ▶ Cannot find all consequences in polynomial time [S'03,FS'02,T'03]

• Example:

\[
4x_1 + 3x_2 + 2x_3 = 5
\]

  – Replace by \((4x_1 + 3x_2 + 2x_3 \geq 5) \land (4x_1 + 3x_2 + 2x_3 \leq 5)\)
More on PB Constraints

• How about $\sum_{j=1}^{n} a_j x_j = k$?
  – Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    ▶ $\sum_{j=1}^{n} a_j x_j = k$ is a subset-sum constraint
      (special case of a knapsack constraint)
    ▶ Cannot find all consequences in polynomial time

[S'03,FS'02,T'03]

• Example:

  $4x_1 + 3x_2 + 2x_3 = 5$

  – Replace by $(4x_1 + 3x_2 + 2x_3 \geq 5) \land (4x_1 + 3x_2 + 2x_3 \leq 5)$
  – Let $x_2 = 0$
More on PB Constraints

- How about \( \sum_{j=1}^{n} a_j x_j = k \)?
  - Can use \((\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)\), but...
    - \( \sum_{j=1}^{n} a_j x_j = k \) is a subset-sum constraint
      (special case of a knapsack constraint)
    - **Cannot** find all consequences in polynomial time

- Example:

  \[
  4x_1 + 3x_2 + 2x_3 = 5
  \]

  - Replace by \((4x_1 + 3x_2 + 2x_3 \geq 5) \land (4x_1 + 3x_2 + 2x_3 \leq 5)\)
  - Let \( x_2 = 0 \)
  - Either constraint can still be satisfied, but **not** both
Outline

Boolean-Based Optimization

Example Applications

Fundamental Techniques

Practical Algorithms

Results, Conclusions & Research Directions
Outline

Boolean-Based Optimization

Example Applications

Fundamental Techniques

Practical Algorithms
  Notation
  B&B Search for MaxSAT & PBO
  Iterative SAT Solving
  Core-Guided Algorithms

Results, Conclusions & Research Directions
Definitions

- **Cost of assignment:**
  - Sum of weights of unsatisfied clauses

- **Optimum solution (OPT):**
  - Assignment with minimum cost

- **Upper Bound (UB):**
  - Assignment with cost not less than OPT
  - E.g. \( \sum_{c_i \in \varphi} w_i + 1 \); hard clauses may be inconsistent

- **Lower Bound (LB):**
  - No assignment with cost no larger than LB
  - E.g. \(-1\); it may be possible to satisfy all soft clauses
Definitions

- **Cost of assignment:**
  - Sum of weights of *unsatisfied* clauses

- **Optimum solution (OPT):**
  - Assignment with *minimum* cost

- **Upper Bound (UB):**
  - Assignment with cost *not less* than OPT
  - E.g. $\sum_{c_i \in \varphi} w_i + 1$; hard clauses may be inconsistent

- **Lower Bound (LB):**
  - No assignment with cost *no larger* than LB
  - E.g. $-1$; it may be possible to satisfy all soft clauses

![Diagram showing relationships between OPT, LB, and UB]
Outline

Boolean-Based Optimization

Example Applications

Fundamental Techniques

Practical Algorithms
  Notation
  B&B Search for MaxSAT & PBO
  Iterative SAT Solving
  Core-Guided Algorithms

Results, Conclusions & Research Directions
Branch-and-Bound Search for MaxSAT

- Unit propagation is unsound for MaxSAT

\[ \{\{x_1\}, \{\neg x_1, \neg x_2\}, \{\neg x_1, \neg x_3\}, \{x_2\}, \{x_3\}\} \]
Branch-and-Bound Search for MaxSAT

- Unit propagation is **unsound** for MaxSAT [e.g. BLM’07]

\[
\{\{x_1\}, \{\neg x_1, \neg x_2\}, \{\neg x_1, \neg x_3\}, \{x_2\}, \{x_3\}\}
\]

- Standard B&B search [LMP’07,HLO’08,LHG’08]
  - **No** unit propagation
    - **No** conflict-driven clause learning
Branch-and-Bound Search for MaxSAT

• Unit propagation is **unsound** for MaxSAT [e.g. BLM’07]

\[
\{\{x_1\}, \{-x_1, -x_2\}, \{-x_1, -x_3\}, \{x_2\}, \{x_3\}\}
\]

• Standard B&B search [LMP’07,HLO’08,LHG’08]
  – **No** unit propagation
    ▶ **No** conflict-driven clause learning

• Refine **UBs** on number of empty clauses

• Estimate **LBs**
  – Unit propagation provides LBs
  – Bound search when \( \text{LB} \geq \text{UB} \)

• Inference rules to prune search [HL’06,LMP’07]

• Optionally: use stochastic local search to identify UBs [HLO’08]
Branch-and-Bound Search for PBO

\[
\begin{align*}
\text{minimize} & \quad \sum_{j \in N} w_j \cdot x_j \\
\text{subject to} & \quad \sum_{j \in N} a_{ij} l_j \geq b_i, \\
& \quad l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0, 1\}, a_{ij}, b_i, w_j \in \mathbb{N}_0^+.
\end{align*}
\]

- Standard B&B search
- Refine UBs on value of cost function
  - Any model for the constraints refines UB
- Estimate LBs
  - Standard techniques: LP relaxations; MIS; etc.
  - Bound search when LB ≥ UB
- Native handling of PB constraints (optional)
Branch-and-Bound Search for PBO

\[
\begin{align*}
\text{minimize} & \quad \sum_{j \in N} w_j \cdot x_j \\
\text{subject to} & \quad \sum_{j \in N} a_{ij} l_j \geq b_i, \\
& \quad l_j \in \{x_j, \bar{x}_j\}, x_j \in \{0, 1\}, a_{ij}, b_i, w_j \in \mathbb{N}_0^+ \end{align*}
\]

- Standard B&B search
- Refine UBs on value of cost function
  - Any model for the constraints refines UB
- Estimate LBs
  - Standard techniques: LP relaxations; MIS; etc.
  - Bound search when LB \geq UB
- Native handling of PB constraints (optional)
- Integrate SAT techniques
  - Unit propagation; Clause learning; Restarts; VSIDS; etc.

[MMS'02, MMS'04, MMS'06, SS'06, NO'06]
Outline

Boolean-Based Optimization

Example Applications

Fundamental Techniques

Practical Algorithms
  Notation
  B&B Search for MaxSAT & PBO
  Iterative SAT Solving
  Core-Guided Algorithms

Results, Conclusions & Research Directions
Iterative SAT Solving

- Iteratively refine upper bound (UB) until UB = OPT
  - Linear search SAT-UNSAT (or strengthening)
Iterative SAT Solving

- Iteratively refine upper bound (UB) until $UB = OPT$
  - Linear search $SAT$-$UNSAT$ (or strengthening)
- Iteratively refine lower bound (LB) until $LB = OPT$
  - Linear search $UNSAT$-$SAT$
Iterative SAT Solving

- Iteratively refine upper bound (UB) until $UB = OPT$
  - Linear search SAT-UNSAT (or strengthening)
- Iteratively refine lower bound (LB) until $LB = OPT$
  - Linear search UNSAT-SAT
- Iteratively refine lower & upper bounds until $LB_k = UB_k - 1$
  - Linear search by refining LB&UB
  - Binary search on cost of unsatisfied clauses
Iterative SAT Solving

- Iteratively refine upper bound (UB) until $UB = OPT$
  - Linear search SAT-UNSAT (or strengthening)

- Iteratively refine lower bound (LB) until $LB = OPT$
  - Linear search UNSAT-SAT

- Iteratively refine lower & upper bounds until $LB_k = UB_k - 1$
  - Linear search by refining LB&UB
  - Binary search on cost of unsatisfied clauses

- By default:
  - All soft clauses relaxed: replace $c_i$ with $c_i \cup \{r_i\}$
  - Cardinality/PB constraints to represent LBs & UBs
Iterative SAT Solving

- Iteratively refine upper bound (UB) until $UB = OPT$
  - Linear search SAT-UNSAT (or strengthening)
- Iteratively refine lower bound (LB) until $LB = OPT$
  - Linear search UNSAT-SAT
- Iteratively refine lower & upper bounds until $LB_k = UB_k - 1$
  - Linear search by refining LB&UB
  - Binary search on cost of unsatisfied clauses

By default: Not for core-guided approaches!
- All soft clauses relaxed: replace $c_i$ with $c_i \cup \{r_i\}$
- Cardinality/PB constraints to represent LBs & UBs
Iterative SAT Solving – Refine UB

\[ \text{LB} \rightarrow \text{OPT} \rightarrow \text{UB}_0 \]

- Require \( \sum w_i r_i \leq UB_0 - 1 \)
Iterative SAT Solving – Refine UB

- Require $\sum_w w_i r_i \leq UB_0 - 1$
- While SAT, refine UB
  - New UB given by cost of unsatisfied clauses, i.e. $\sum w_i r_i$
Iterative SAT Solving – Refine UB

- Require $\sum w_i r_i \leq UB_0 - 1$
- While SAT, refine UB
  - New UB given by cost of unsatisfied clauses, i.e. $\sum w_i r_i$
Iterative SAT Solving – Refine UB

- Require $\sum w_i r_i \leq UB_0 - 1$
- While $SAT$, refine $UB$
  - New $UB$ given by cost of unsatisfied clauses, i.e. $\sum w_i r_i$
- Repeat until constraint $\sum w_i r_i \leq UB_k - 1$ becomes $UNSAT$
  - $UB_k$ denotes the optimum value
Iterative SAT Solving – Refine UB

- Require $\sum w_i r_i \leq UB_0 - 1$
- While SAT, refine UB
  - New UB given by cost of unsatisfied clauses, i.e. $\sum w_i r_i$
- Repeat until constraint $\sum w_i r_i \leq UB_k - 1$ becomes UNSAT
  - $UB_k$ denotes the optimum value

- Worst-case # of iterations exponential on instance size
Iterative SAT Solving – Refine UB

- Require $\sum w_i r_i \leq UB_0 - 1$
- While SAT, refine UB
  - New UB given by cost of unsatisfied clauses, i.e. $\sum w_i r_i$
- Repeat until constraint $\sum w_i r_i \leq UB_k - 1$ becomes UNSAT
  - $UB_k$ denotes the optimum value

- Worst-case # of iterations exponential on instance size

- Example tools:
  - Minisat+: CNF encoding of constraints
  - SAT4J: native handling of constraints
  - QMaxSat: CNF encoding of constraints
  - ...
Iterative SAT Solving – Refine LB

- Require \( \sum w_i r_i \leq LB_0 + 1 \)
Iterative SAT Solving – Refine LB

• Require $\sum w_i r_i \leq LB_0 + 1$
• While UNSAT, refine LB, i.e. $LB_k \leftarrow LB_{k-1} + 1$
Iterative SAT Solving – Refine LB

- Require $\sum w_i r_i \leq LB_0 + 1$
- While UNSAT, refine LB, i.e. $LB_k \leftarrow LB_{k-1} + 1$
Iterative SAT Solving – Refine LB

- Require $\sum w_i r_i \leq LB_0 + 1$
- While UNSAT, refine LB, i.e. $LB_k \leftarrow LB_{k-1} + 1$
- Repeat until constraint $\sum w_i r_i \leq LB_k$ becomes SAT
  - $LB_k$ denotes the optimum value
Iterative SAT Solving – Refine LB

- Require $\sum w_i r_i \leq LB_0 + 1$
- While UNSAT, refine LB, i.e. $LB_k \leftarrow LB_{k-1} + 1$
- Repeat until constraint $\sum w_i r_i \leq LB_k$ becomes SAT
  - $LB_k$ denotes the optimum value
- Worst-case # of iterations exponential on instance size
Iterative SAT Solving – Refine LB

- Require $\sum w_i r_i \leq LB_0 + 1$
- While UNSAT, refine LB, i.e. $LB_k \leftarrow LB_{k-1} + 1$
- Repeat until constraint $\sum w_i r_i \leq LB_k$ becomes SAT
  - $LB_k$ denotes the optimum value

- Worst-case # of iterations exponential on instance size

- Example tools:
  - No known implementations. Why?
  - Some core-guided MaxSAT solvers
    - But policies for updating LB are different
    - Unclear whether worst-case # of iterations remains exponential
Iterative SAT Solving – Binary Search

• Invariant: $\text{LB}_k \leq \text{UB}_k - 1$
• Require $\sum w_i r_i \leq m_0$

$m_0 = \lfloor (\text{LB}_0 + \text{UB}_0)/2 \rfloor$
Iterative SAT Solving – Binary Search

- Invariant: $LB_k \leq UB_k - 1$
- Require $\sum w_i r_i \leq m_0$
- Repeat
  - If UNSAT, refine $LB_1 = m_0, \ldots$
  - Compute new mid value $m_1, \ldots$
Iterative SAT Solving – Binary Search

- Invariant: $LB_k \leq UB_k - 1$
- Require $\sum w_i r_i \leq m_0$
- Repeat
  - If UNSAT, refine $LB_1 = m_0, \ldots$
  - Compute new mid value $m_1, \ldots$
  - If SAT, refine $UB_3 = m_2, \ldots$
Iterative SAT Solving – Binary Search

- Invariant: \( LB_k \leq UB_k - 1 \)
- Require \( \sum w_i r_i \leq m_0 \)
- Repeat
  - If UNSAT, refine \( LB_1 = m_0, \ldots \)
  - Compute new mid value \( m_1, \ldots \)
  - If SAT, refine \( UB_3 = m_2, \ldots \)
Iterative SAT Solving – Binary Search

- Invariant: $\text{LB}_k \leq \text{UB}_k - 1$
- Require $\sum w_i r_i \leq m_0$
- Repeat
  - If UNSAT, refine $\text{LB}_1 = m_0, \ldots$
  - Compute new mid value $m_1, \ldots$
  - If SAT, refine $\text{UB}_3 = m_2, \ldots$
- Until $\text{LB}_k = \text{UB}_k - 1$
Iterative SAT Solving – Binary Search

- Invariant: \( \text{LB}_k \leq \text{UB}_k - 1 \)
- Require \( \sum w_i r_i \leq m_0 \)
- Repeat
  - If UNSAT, refine \( \text{LB}_1 = m_0, \ldots \)
  - Compute new mid value \( m_1, \ldots \)
  - If SAT, refine \( \text{UB}_3 = m_2, \ldots \)
- Until \( \text{LB}_k = \text{UB}_k - 1 \)
- Worst-case # of iterations \textbf{linear} on instance size
Iterative SAT Solving – Binary Search

- Invariant: $LB_k \leq UB_k - 1$
- Require $\sum w_i r_i \leq m_0$
- Repeat
  - If UNSAT, refine $LB_1 = m_0, \ldots$
  - Compute new mid value $m_1, \ldots$
  - If SAT, refine $UB_3 = m_2, \ldots$
- Until $LB_k = UB_k - 1$
- Worst-case # of iterations linear on instance size
- Example tools:
  - Counter-based MaxSAT solver
  - MathSAT
  - MSUnCore

[FM'06]
[CFGSS'10]
[HMMS'11]
Outline

Boolean-Based Optimization

Example Applications

Fundamental Techniques

**Practical Algorithms**
- Notation
- B&B Search for MaxSAT & PBO
- Iterative SAT Solving
- Core-Guided Algorithms

Results, Conclusions & Research Directions
What are Core-Guided MaxSAT Algorithms?

- **Drawbacks of iterative SAT solving**
  - All soft clauses are relaxed
    - Number of soft clauses can be large
  - PB/cardinality constraints with large number of variables and (possibly) large rhs
    - Can result in large CNF encodings

- **Core-guided MaxSAT algorithms** use unsatisfiable cores for:
  - Relax soft clauses on demand, i.e. relax clauses only when needed, or
  - Relax all soft clauses, but use unsatisfiable cores for creating simpler PB/cardinality constraints
Many Core-Guided MaxSAT Algorithms

- (W)MSU1.X/WPM1
- (W)MSU3
- (W)MSU4
- (W)PM2
- Core-guided binary search (w/ disjoint cores)
  - Bin-Core, Bin-Core-Dis, Bin-Core-Dis2
Many Core-Guided MaxSAT Algorithms

- Algorithms:
  - (W)MSU1.X/WPM1 [FM’06, MSM’08, MMSP’09, ABL’09a]
  - (W)MSU3 [MSP’07]
  - (W)MSU4 [MSP’08]
  - (W)PM2 [ABL’09a, ABL’09b, ABL’10]
  - Core-guided binary search (w/ disjoint cores)
    - Bin-Core, Bin-Core-Dis, Bin-Core-Dis2

- Other properties:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Type</th>
<th>Relaxation Vars p/ Clause</th>
<th>On Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>(W)MSU1.X/WPM1</td>
<td>UNSAT-SAT</td>
<td>Multiple</td>
<td>Y</td>
</tr>
<tr>
<td>(W)MSU3</td>
<td>UNSAT-SAT</td>
<td>Single</td>
<td>Y</td>
</tr>
<tr>
<td>(W)MSU4</td>
<td>Refine LB&amp;UB</td>
<td>Single</td>
<td>Y</td>
</tr>
<tr>
<td>(W)PM2</td>
<td>UNSAT-SAT</td>
<td>Single</td>
<td>N/Y</td>
</tr>
<tr>
<td>Bin-Core</td>
<td>Bin Search</td>
<td>Single</td>
<td>Y</td>
</tr>
<tr>
<td>Bin-Core-Dis</td>
<td>Bin Search</td>
<td>Single</td>
<td>Y</td>
</tr>
<tr>
<td>Bin-Core-Dis2</td>
<td>Bin Search</td>
<td>Single</td>
<td>Y</td>
</tr>
</tbody>
</table>
An Example: (W)MSU1.X

Example CNF formula

\[
\begin{align*}
&x_6 \lor x_2 & &\neg x_6 \lor x_2 & &\neg x_2 \lor x_1 & &\neg x_1 \\
&\neg x_6 \lor x_8 & &x_6 \lor \neg x_8 & &x_2 \lor x_4 & &\neg x_4 \lor x_5 \\
&x_7 \lor x_5 & &\neg x_7 \lor x_5 & &\neg x_5 \lor x_3 & &\neg x_3
\end{align*}
\]
An Example: (W)MSU1.X

\[ x_6 \lor x_2 \quad \neg x_6 \lor x_2 \]
\[ \neg x_6 \lor x_8 \quad x_6 \lor \neg x_8 \]
\[ x_7 \lor x_5 \quad \neg x_7 \lor x_5 \]
\[ \neg x_2 \lor x_1 \quad \neg x_1 \]
\[ x_2 \lor x_4 \quad \neg x_4 \lor x_5 \]
\[ \neg x_5 \lor x_3 \quad \neg x_3 \]

Formula is **UNSAT**; \( \text{OPT} \leq |\varphi| - 1 \); Get unsat core
An Example: (W)MSU1.X

\[
\begin{align*}
    & x_6 \vee x_2 & & \neg x_6 \vee x_2 & & \neg x_2 \vee x_1 \vee r_1 & & \neg x_1 \vee r_2 \\
    & \neg x_6 \vee x_8 & & x_6 \vee \neg x_8 & & x_2 \vee x_4 \vee r_3 & & \neg x_4 \vee x_5 \vee r_4 \\
    & x_7 \vee x_5 & & \neg x_7 \vee x_5 & & \neg x_5 \vee x_3 \vee r_5 & & \neg x_3 \vee r_6 \\
\end{align*}
\]

\[\sum_{i=1}^{6} r_i \leq 1\]

Add relaxation variables and AtMost1 constraint
An Example: (W)MSU1.X

\[
\begin{align*}
& x_6 \lor x_2 & & \neg x_6 \lor x_2 & & \neg x_2 \lor x_1 \lor r_1 & & \neg x_1 \lor r_2 \\
& \neg x_6 \lor x_8 & & x_6 \lor \neg x_8 & & x_2 \lor x_4 \lor r_3 & & \neg x_4 \lor x_5 \lor r_4 \\
& x_7 \lor x_5 & & \neg x_7 \lor x_5 & & \neg x_5 \lor x_3 \lor r_5 & & \neg x_3 \lor r_6 \\
\sum_{i=1}^{6} r_i & \leq 1
\end{align*}
\]

Formula is (again) **UNSAT**; **OPT \leq |\varphi| - 2**; Get unsat core
An Example: (W)MSU1.X

\[ x_6 \lor x_2 \lor r_7 \quad \neg x_6 \lor x_2 \lor r_8 \quad \neg x_2 \lor x_1 \lor r_1 \lor r_9 \quad \neg x_1 \lor r_2 \lor r_{10} \]

\[ \neg x_6 \lor x_8 \quad x_6 \lor \neg x_8 \quad x_2 \lor x_4 \lor r_3 \quad \neg x_4 \lor x_5 \lor r_4 \]

\[ x_7 \lor x_5 \lor r_{11} \quad \neg x_7 \lor x_5 \lor r_{12} \quad \neg x_5 \lor x_3 \lor r_5 \lor r_{13} \quad \neg x_3 \lor r_6 \lor r_{14} \]

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

Add new relaxation variables and AtMost1 constraint
An Example: (W)MSU1.X

\[
\begin{align*}
  x_6 \lor x_2 \lor r_7 & \quad \neg x_6 \lor x_2 \lor r_8 & \quad \neg x_2 \lor x_1 \lor r_1 \lor r_9 & \quad \neg x_1 \lor r_2 \lor r_{10} \\
  \neg x_6 \lor x_8 & \quad x_6 \lor \neg x_8 & \quad x_2 \lor x_4 \lor r_3 & \quad \neg x_4 \lor x_5 \lor r_4 \\
  x_7 \lor x_5 \lor r_{11} & \quad \neg x_7 \lor x_5 \lor r_{12} & \quad \neg x_5 \lor x_3 \lor r_5 \lor r_{13} & \quad \neg x_3 \lor r_6 \lor r_{14} \\
  \sum_{i=1}^{6} r_i \leq 1 & \quad \sum_{i=7}^{14} r_i \leq 1
\end{align*}
\]

Instance is now SAT
An Example: (W)MSU1.X

\[
\begin{align*}
    x_6 \lor x_2 \lor r_7 & \quad \neg x_6 \lor x_2 \lor r_8 & \quad \neg x_2 \lor x_1 \lor r_1 \lor r_9 & \quad \neg x_1 \lor r_2 \lor r_{10} \\
    \neg x_6 \lor x_8 & \quad x_6 \lor \neg x_8 & \quad x_2 \lor x_4 \lor r_3 & \quad \neg x_4 \lor x_5 \lor r_4 \\
    x_7 \lor x_5 \lor r_{11} & \quad \neg x_7 \lor x_5 \lor r_{12} & \quad \neg x_5 \lor x_3 \lor r_5 \lor r_{13} & \quad \neg x_3 \lor r_6 \lor r_{14} \\
    \sum_{i=1}^{6} r_i \leq 1 & \quad \sum_{i=7}^{14} r_i \leq 1
\end{align*}
\]

MaxSAT solution is $|\varphi| - I = 12 - 2 = 10$
Organization of MSU1.X

• Clauses characterized as:
  – **Soft**: initial set of soft clauses
  – **Hard**: initially hard, or added during execution of algorithm
    ▶ E.g. clauses from AtMost1 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

• (Partial) MaxSAT solution is $|\varphi| - I$
  – $I$: number of iterations ($\equiv$ number of computed unsat cores)
Organization of MSU1.X

- Clauses characterized as:
  - **Soft**: initial set of soft clauses
  - **Hard**: initially hard, or added during execution of algorithm
    ▶ E.g. clauses from AtMost1 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

- (Partial) MaxSAT solution is $|\varphi| - \mathcal{I}$
  - $\mathcal{I}$: number of iterations ($\equiv$ number of computed unsat cores)

- Can be adapted for weighted MaxSAT
  [FM’06]
  [ABL’09a,MMSP’09]
Binary Search For MaxSAT (Bin)

\[(R, \varphi_W) \leftarrow \text{Relax}(\emptyset, \varphi, \text{Soft}(\varphi))\]
\[(\lambda, \mu, A_M) \leftarrow (-1, \sum_{i=1}^m w_i + 1, \emptyset)\]

while \(\lambda < \mu - 1\) do

\[\nu \leftarrow \lfloor (\lambda + \mu)/2 \rfloor\]
\[\varphi_E \leftarrow \text{CNF}(\sum_{r_i \in R} w_i r_i \leq \nu)\]
\[(st, A) \leftarrow \text{SAT}(\varphi_W \cup \varphi_E)\]

if \(st = \text{true}\) then

\[(A_M, \mu) \leftarrow (A, \sum_{i=1}^m w_i A\langle r_i \rangle)\]
else

\[\lambda \leftarrow \nu\]

return \(\text{Init}(A_M)\)
Core-Guided Binary Search (Bin-Core)

\[(R, \varphi_W, \varphi_S) \leftarrow (\emptyset, \varphi, \text{Soft}(\varphi))\]
\[(\lambda, \mu, A_M) \leftarrow (-1, \sum_{i=1}^m w_i + 1, \emptyset)\]

while \(\lambda < \mu - 1\) do

\[\nu \leftarrow \lfloor (\lambda + \mu) / 2 \rfloor\]
\[\varphi_E \leftarrow \text{CNF}(\sum_{r_i \in R} w_i r_i \leq \nu)\]
\[(\text{st}, \varphi_C, A) \leftarrow \text{SAT}(\varphi_W \cup \varphi_E)\]

if st = true then

\[(A_M, \mu) \leftarrow (A, \sum_{i=1}^m w_i A\langle r_i \rangle)\]
else

if \(\varphi_C \cap \varphi_S = \emptyset\) then

\[\lambda \leftarrow \nu\]
else

\[(R, \varphi_W) \leftarrow \text{Relax}(R, \varphi_W, \varphi_C \cap \varphi_S)\]

return \(\text{Init}(A_M)\)
Bin-Core with Disjoint Cores (Bin-Core-Dis)

- Organization similar to Bin-Core

- Keep set of disjoint unsatisfiable cores
  - Need to join unsatisfiable cores  
    [HMMS’11]

- Integrate lower & upper bounds
  - Essential to reduce number of iterations  
    [HMMS’11,MHMS’12]

- Integrate additional pruning techniques  
  [MHMS’12]
Outline

- Boolean-Based Optimization
- Example Applications
- Fundamental Techniques
- Practical Algorithms
- Results, Conclusions & Research Directions
Results for Industrial & Crafted Instances (2011)
(Recent) Results for Non-Random Instances 2009–2011

2012 Results

- bin-c-d2
- bin-c-d
- bin-c
- minimaxsat
- bin
- sat4j-maxsat
- wpm2v2
- wpm2v1
- wpm1
- msc_msu1
- pm2
Conclusions

• Equivalence between Boolean optimization representations
  – Pseudo-Boolean Optimization (PBO) (or 0-1 ILP)
  – Maximum Satisfiability (MaxSAT)
  – etc.

• Overview of SAT-based Boolean optimization algorithms
  – B&B PBO
  – B&B MaxSAT
  – Iterative SAT solving
  – Core-guided MaxSAT

• Core-guided algorithms exhibit (moderate) performance edge
  – Disclaimer: Industrial & crafted instances from MaxSAT evaluations
Research Directions

- Core-guided MaxSAT algorithms
  - More algorithms?
  - Can we do better than core-guided binary search?
  - Theoretical analysis?
    - Worst-case $\#$ of iterations?

- MaxSAT vs. MaxSMT
  - Can use the same algorithms

- MaxSAT vs. MaxCSP
  - Effective alternative?

- MaxSAT vs. ILP
  - Complementary approaches?

- More practical applications
  - Recent examples
    - Error localization in C code
    - Reasoning over biological networks
  - Practical applications drive development of efficient algorithms
Thank You
<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Title</th>
<th>Conference/Journal</th>
<th>Pages/Publication Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>B'68</td>
<td>K. Batcher</td>
<td>Sorting Networks and Their Applications</td>
<td>AFIPS Spring Joint Computing Conference 1968</td>
<td>307-314</td>
</tr>
<tr>
<td>FP'01</td>
<td>A. Frisch, T. Peugniez</td>
<td>Solving Non-Boolean Satisfiability Problems with Stochastic Local Search</td>
<td>IJCAI 2001</td>
<td>282-290</td>
</tr>
<tr>
<td>FS'02</td>
<td>T. Fahle, M. Sellmann</td>
<td>Cost Based Filtering for the Constrained Knapsack Problem</td>
<td>Annals OR 115(1-4)</td>
<td>73-93 (2002)</td>
</tr>
<tr>
<td>S'03</td>
<td>M. Sellmann</td>
<td>Approximated Consistency for Knapsack Constraints</td>
<td>CP 2003</td>
<td>679-693</td>
</tr>
<tr>
<td>S'05</td>
<td>C. Sinz</td>
<td>Towards an Optimal CNF Encoding of Boolean Cardinality Constraints</td>
<td>CP 2005</td>
<td>827-831</td>
</tr>
<tr>
<td>Reference Set</td>
<td>Authors and Title</td>
<td>Details</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>---------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BBR’09</td>
<td>O. Bailleux, Y. Boufkhad, O. Roussel: New Encodings of Pseudo-Boolean Constraints into CNF.</td>
<td>SAT 2009: 181-194</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CZI’10</td>
<td>M. Codish, M. Zason-Ivry: Pairwise Cardinality Networks.</td>
<td>LPAR (Dakar) 2010: 154-172</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title and Details</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMP’05</td>
<td>C.-M. Li, F. Manya, J. Planes</td>
<td>Exploiting Unit Propagation to Compute Lower Bounds in Branch and Bound Max-SAT Solvers. CP 2005: 403-414</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HL’06</td>
<td>F. Heras, J. Larrosa</td>
<td>New Inference Rules for Efficient Max-SAT Solving. AAAI 2006</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Authors and Details</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MMS'00</td>
<td>V. Manquinho, J. Marques-Silva: Search Pruning Conditions for Boolean Optimization. ECAI 2000: 103-107</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CK'03</td>
<td>D. Chai, A. Kuehlmann: A fast pseudo-boolean constraint solver. DAC 2003: 830-835</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
<td>Journal/Conference</td>
<td>Pages/Volume</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Year</td>
<td>Authors</td>
<td>Title</td>
<td>Journal/Conference</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>-----------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>NO’06</td>
<td>R. Nieuwenhuis, A. Oliveras</td>
<td>On SAT Modulo Theories and Optimization Problems.</td>
<td>SAT 2006: 156-169</td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Title</td>
<td>Conference/Journal</td>
<td>Pages/Volume</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------------------------------------------------------</td>
<td>-----------------------------</td>
<td>--------------</td>
<td></td>
</tr>
<tr>
<td>FM’06</td>
<td>Z. Fu, S. Malik: On Solving the Partial MAX-SAT Problem</td>
<td>SAT 2006</td>
<td>252-265</td>
<td></td>
</tr>
<tr>
<td>MSP’08</td>
<td>J. Marques-Silva, Jordi Planes: Algorithms for Maximum Satisfiability using Unsatisfiable Cores</td>
<td>DATE 2008: 408-413</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABL’09a</td>
<td>C. Ansotegui, M. Bonet, J. Levy: Solving (Weighted) Partial MaxSAT through Satisfiability Testing</td>
<td>SAT 2009: 427-440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABL’10</td>
<td>C. Ansotegui, M. Bonet, J. Levy: A New Algorithm for Weighted Partial MaxSAT. AAAI 2010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Author(s)</td>
<td>Title and Details</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------</td>
<td>------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSW’03</td>
<td>M. G. de la Banda, P. J. Stuckey, J. Wazny</td>
<td>finding all minimal unsatisfiable subsets. PPDP 2003: 32-43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS’05</td>
<td>J. Bailey, P. J. Stuckey</td>
<td>Discovery of Minimal Unsatisfiable Subsets of Constraints Using Hitting Set Dualization. PADL 2005: 174-186</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reference</td>
<td>Title</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AL’08</td>
<td>J. Argelich, I. Lynce: CNF Instances from the Software Package Installation Problem. RCRA 2008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>References</td>
<td>Authors</td>
<td>Title</td>
<td>Conference</td>
<td>Year</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-------</td>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>AN’10</td>
<td>R. Asin, R. Nieuwenhuis</td>
<td>Curriculum-based Course Timetabling with SAT and MaxSAT</td>
<td>PATAT</td>
<td>2010</td>
</tr>
<tr>
<td>GLMSO’10</td>
<td>A. Graca, I. Lynce, J. Marques-Silva, and A. Oliveira</td>
<td>Efficient and Accurate Haplotype Inference by Combining Parsimony and Pedigree Information</td>
<td>ANB</td>
<td>2010</td>
</tr>
<tr>
<td>AVFPS’10</td>
<td>C. Ardagna, S. Vimercati, S. Foresti, S. Paraboschi, P. Samarati</td>
<td>minimizing disclosure of private information in credential-based interactions: a graph-based approach</td>
<td>SocialCom/PASSAT</td>
<td>2010</td>
</tr>
</tbody>
</table>
References – Applications III


JM’11  M. Jose, R. Majumdar: Cause clue clauses: error localization using maximum satisfiability. PLDI 2011: 437-446

GL’12  J. Guerra, I. Lynce: Reasoning over Biological Networks using Maximum Satisfiability. CP 2012