A BDD based Pseudo Boolean Constraint Solver

Valentin Christian Johannes Kaspar Mayer-Eichberger

Universidade Nova de Lisboa, FCT

Presentation at EMCL Workshop, 22.02.2008
Outline

1. Pseudo Boolean Constraints
2. Binary Decision Diagrams
3. Transformation from PBCs to BDDs
4. Create Consistency between Constraints: Conjunction
5. Ideas for search: unit propagation, no-good learning, heuristics
6. Preliminary Results
Quote from A.E.

Understanding vs. Explanation

You do not really understand something unless you can explain it to your grandmother.
Linear Pseudo Boolean Constraints

**PBC**

\[ \sum a_i x_i \geq t \text{ with } a_i, t \in \mathbb{Z} \text{ and } x_i \in \{0, 1\} \]

- example: \(32x_1 + 16x_2 + 8x_3 + 4x_4 + 2x_5 + x_6 \geq 20\)
- also known as threshold functions or 0/1 Integer Programming
- Why better than CNF?
  - clauses
    - example: \((x_1 \lor \bar{x_2} \lor x_3) \equiv x_1 + \bar{x_2} + x_3 \geq 1\)
  - cardinality (or M of N)
    - example: \(x_1 + x_2 + x_3 + \ldots + x_N \geq M\)
  - reification
    - \((\sum a_i x_i \geq t) \Leftrightarrow y \text{ and } M = \sum |a_i| + |t|\)
    - \((\sum a_i x_i - M y < t) \land (\sum a_i x_i + M \bar{y} \geq t)\)
Personal Slide for MITO

\[ C = \{ krr, agents, \ldots, constraintFinite, \ldots, \} \]
\[ M = \{ it, krai, krragents, constraints, semanticweb, tscl \ldots \} \]
\[ S = \{ S_1, S_2, \ldots \} \]
\[ crs : (\hat{C} \cup \hat{M}) \rightarrow \mathbb{Z}^+ \]

- \( \left( \sum c \cdot crs(\hat{c}) \right) + crs(project) + crs(foundations) + crs(thesis) \geq 120 \)
- \( \sum_{m \in M} m \geq 3 \)
- \( \left( \sum_{c \in m} c \cdot crs(\hat{c}) \geq crs(\hat{m}) \right) \iff m \)
- \( \sum_{c \notin s} -s \cdot c \geq 0 \)
- \( \bigwedge_{\hat{m}_1 \cap \hat{m}_2 \neq \emptyset} \left( (m_1 \land m_2) \rightarrow (\sum_{c \in (m_1 \cap m_2)} c \cdot crs(\hat{c}) \geq crs(\hat{m}_1) + crs(\hat{m}_2)) \right) \)
- \( \min \sum c \cdot crs(\hat{c}) + crs(project) + crs(foundations) \)
Boolean Decision Diagrams

- Graph based representation for boolean functions [Bry86]
- Research the last 20 years, mainly synthesis of circuits and formal verification
- ROBDD = Reduced Ordered BDDs
  - Fixed variable ordering
  - Merge common subtrees
  - Delete redundant nodes
What are they good for

- uniqueness
  - a node stores only (id, level, zero, one)
  - every function is pointer to a node in one big BDD
  - every boolean (sub) function occurs exactly once

- nice algorithms:
  - satisfiability ($O(1)$)
  - negation ($O(1)$)
  - equality ($O(1)$)
  - and, or ... ($O(n \times m)$)
  - sat-count ($O(n)$)

- order strong influence on size
Example: ThresholdBDDs

\[ x \leq 49 \quad x \geq 20 \quad x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3 \]
**Example:**  

\[ 4x_0 + 3x_1 + 4x_2 + 4x_3 + 6x_4 \geq 9 \]

**bounds updating**

\[
ub = \min(ub_0, ub_1 - \text{weight}) \\
lb = \max(lb_0, lb_1 - \text{weight})
\]

\[
x_n \quad [lb_0, ub_0] \\
\]

\[
x_{n+1} \quad [lb_1, ub_1]
\]
Theoretical Result for Threshold BDDs [KYS94]

- **output sensitive** algorithm [Beh07, ES06]
- polynomial weights ... good news
  - polynomial size
  - independent of order
  - example: $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3$
  - $\sum_{i=1}^{n} i^2 x_i \geq (n/2)^2$
- exponential weights ... depends
  - might be small
  - or depends on order
    - example: $15x_1 + 14x_2 + 12x_3 + 8x_4 + 8x_5 + 4x_6 + 2x_7 + 1x_8 \geq 32$
    - $\sum_{i=1}^{n} 2^i x_i \geq 2^{(n/2)}$
  - or always exponential
Example 1 for And Algorithm

- **$C_1$**
  - $x_1$, $x_2$, $x_3$, $x_4$, $x_5$
  - $x_1$ as root
  - $x_2$ and $x_3$ as children of $x_1$
  - $x_4$ and $x_5$ as children of $x_3$
  - $0$ and $1$ as leaves

- **$C_2$**
  - $x_1$, $x_2$, $x_3$, $x_4$, $x_5$
  - $x_1$ as root
  - $x_2$ and $x_3$ as children of $x_1$
  - $x_4$ and $x_5$ as children of $x_3$
  - $0$ and $1$ as leaves

- **$C_1 \land C_2$**
  - $x_1$, $x_2$, $x_3$, $x_4$, $x_5$
  - $x_1$ as root
  - $x_2$ and $x_3$ as children of $x_1$
  - $x_4$ and $x_5$ as children of $x_3$
  - $0$ and $1$ as leaves

A BDD based Pseudo Boolean Constraint Solver
Example 2 for And Algorithm

\[ C_1 \land C_2 \]

\[ C_3 \]

\[ C_1 \land C_2 \land C_3 \]
Clustering [DK03]

A BDD based Pseudo Boolean Constraint Solver
Unit Propagation Heuristics for Search

- **unit propagation**
  - determine if \( \text{bdd } g \Rightarrow x_i = 0/1 \)
  - update the other bdds

- **conflict learning (nogoods)**
  - find minimal nogood is NP-complete [Sub08]

- **heuristics: which variable, which value?**
  - useful information on every node
  - model count (SAT count) (probability of success)
  - size of the bdd (how complex)
  - shortest distance to 0/1 sink (fast fail, fast don’t care)
  - number of shared nodes (how special)
No Goods

A BDD based Pseudo Boolean Constraint Solver
until now only seen toy examples...

**Pseudo Boolean Evaluation**

- yearly held competition under SAT Conference
- 2251 benchmark problems from 3 years 2005-07, next in 2009
- last year 16 solvers, mostly tuned SAT solvers
- categories:
  - SMALLINT/BIGINT
  - SATUNSAT/OPT
  - LIN/NONLINEAR
- number factorization, graph problems, minimum-size prime implicant, nqueens, pigeon hole, traveling tournament/salesman, knapsack, industrial problems
- problems with up to 200,000 variables and 100,000 constraints
A BDD based Pseudo Boolean Constraint Solver

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Var/Cons</th>
<th>Solutions</th>
<th>Time(J) in sec</th>
<th>Time(P) in sec</th>
<th>size in 10^5</th>
<th>Others succ./all</th>
<th>Best in sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>pig-cl-11-10</td>
<td>110/561</td>
<td>0</td>
<td>0.25</td>
<td>8.52</td>
<td>0.8</td>
<td>9/16</td>
<td>8.0</td>
</tr>
<tr>
<td>pig-cl-15-10</td>
<td>150/1065</td>
<td>0</td>
<td>0.53</td>
<td>4.52</td>
<td>0.4</td>
<td>9/16</td>
<td>17.9</td>
</tr>
<tr>
<td>pig-ca-11-10</td>
<td>110/21</td>
<td>0</td>
<td>0.47</td>
<td>2.08</td>
<td>0.1</td>
<td>12/16</td>
<td>0.0</td>
</tr>
<tr>
<td>pig-ca-15-10</td>
<td>150/25</td>
<td>0</td>
<td>0.44</td>
<td>4.50</td>
<td>0.1</td>
<td>11/16</td>
<td>0.0</td>
</tr>
<tr>
<td>prob1</td>
<td>72/2</td>
<td>8.59E8</td>
<td>20.5</td>
<td>-</td>
<td>13</td>
<td>4/7</td>
<td>0.06</td>
</tr>
<tr>
<td>prob2</td>
<td>68/2</td>
<td>2.04E6</td>
<td>21.6</td>
<td>-</td>
<td>13</td>
<td>4/7</td>
<td>0.6</td>
</tr>
<tr>
<td>prob3</td>
<td>52/2</td>
<td>0</td>
<td>0.875</td>
<td>1.5h</td>
<td>1.0</td>
<td>0/7</td>
<td>-</td>
</tr>
<tr>
<td>prob4</td>
<td>82/2</td>
<td>6.31E7</td>
<td>10.4</td>
<td>-</td>
<td>79</td>
<td>3/7</td>
<td>254.8</td>
</tr>
<tr>
<td>prob5</td>
<td>80/2</td>
<td>2.17E10</td>
<td>94.0</td>
<td>-</td>
<td>19</td>
<td>5/7</td>
<td>13.9</td>
</tr>
<tr>
<td>prob6</td>
<td>93/2</td>
<td>2.18E5</td>
<td>9.8</td>
<td>-</td>
<td>7.6</td>
<td>4/7</td>
<td>0.1</td>
</tr>
<tr>
<td>prob7</td>
<td>96/2</td>
<td>4.19E12</td>
<td>34.0</td>
<td>-</td>
<td>2.0</td>
<td>1/7</td>
<td>498.8</td>
</tr>
<tr>
<td>prob8</td>
<td>86/2</td>
<td>6.74E6</td>
<td>21.1</td>
<td>-</td>
<td>1.4</td>
<td>1/7</td>
<td>7.5</td>
</tr>
<tr>
<td>cuww1</td>
<td>60/2</td>
<td>1</td>
<td>0.13</td>
<td>0.69</td>
<td>0.01</td>
<td>6/7</td>
<td>0.01</td>
</tr>
<tr>
<td>cuww2</td>
<td>70/2</td>
<td>1</td>
<td>0.14</td>
<td>0.50</td>
<td>0.01</td>
<td>1/7</td>
<td>24.1</td>
</tr>
<tr>
<td>cuww3</td>
<td>68/2</td>
<td>2</td>
<td>0.13</td>
<td>0.50</td>
<td>0.01</td>
<td>1/7</td>
<td>16.5</td>
</tr>
<tr>
<td>cuww4</td>
<td>82/2</td>
<td>1</td>
<td>0.16</td>
<td>1.34</td>
<td>0.03</td>
<td>1/7</td>
<td>15.2</td>
</tr>
<tr>
<td>cuww5</td>
<td>86/2</td>
<td>1</td>
<td>0.17</td>
<td>1.16</td>
<td>0.02</td>
<td>1/7</td>
<td>36.3</td>
</tr>
</tbody>
</table>
Quote from A.E.

**What are we doing?**

*The whole of science is nothing more than a refinement of everyday thinking.*
What is left?

- **finish my thesis!!!!!**
  - implementation of nogoods, unitpropagation
  - solve more benchmark problems, OPT
  - define and evaluate different heuristics
  - use the constraint graph

- **further ideas**
  - incorporate ideas from IP (branch’n bound, relaxations, gomory cuts, gröbner bases, generating functions etc.)
  - determining better variable orderings

- **hints for YOU**
  - don’t become a third year student
  - but do enjoy life in foreign countries
Bibliography

Markus Behle.
On threshold bdds and the optimal variable ordering problem.

R.E. Bryant.
Graph-based algorithms for boolean function manipulation.

Robert Damiano and James Kukula.
Checking satisfiability of a conjunction of bdds.

Niklas Een and Niklas Sörensson.
Translating pseudo-boolean constraints into sat.
2006.

Hosaka Kazuhisa, Takenaga Yasuhiko, and Yamjima Shuzo
Algorithm 1: buildBDD(level, value)

Input: current level \(i\) and temporary sum \(value\)
Output: key to the ROBDD

1. if \(level > total\) then return \(0\)
2. else if \(u\)-tbl.exist\((value, level)\) then return \(u\)-tbl.get\((value, level)\)
3. else if \(value \geq threshold\) then return \(1\)
4. else
5. \(t \leftarrow \text{buildBDD}(level + 1, value + \text{coefficient}[level])\)
6. \(s \leftarrow \text{buildBDD}(level + 1, value)\)
7. \(lb \leftarrow \max(lbs, lb_t - \text{coefficient}[level])\)
8. \(ub \leftarrow \min(ubs, ub_t - \text{coefficient}[level])\)
9. \(\text{return } u\)-tbl.insert\((level, t, s, lb, ub)\)
And Algorithm $f \land g$

**rules**

base cases:

$1 \land f = f$

$0 \land f = 0$

$f \land f = f$

- complexity: $\text{size}(f) \cdot \text{size}(g)$
- tabling of calls
And Algorithm

Algorithm 2: \( \text{and}(f, g) \)

**Input:** two ROBDDS \( f, g \)

**Output:** their conjunction

1. **if** \( f = 0 \) **or** \( g = 0 \) **then return** 0
2. **else if** \( f = 1 \) **then return** \( g \)
3. **else if** \( g = 1 \) **then return** \( f \)
4. **else**
   5. \( k \leftarrow \text{level of } f \)
   6. \( l \leftarrow \text{level of } g \)
   7. \( i \leftarrow \min(k, l) \)
   8. **if** \( k < l \) **then return** \( \text{u-tbl.insert}(i, \text{and}(f_0, g), \text{and}(f_1, g)) \)
   9. **else if** \( l < k \) **then return** \( \text{u-tbl.insert}(i, \text{and}(g_0, f), \text{and}(g_1, f)) \)
10. **else return** \( \text{u-tbl.insert}(i, \text{and}(f_1, g_1), \text{and}(f_0, g_0)) \)
Definitions

Arc-consistency

A variable of a constraint satisfaction problem is arc-consistent with another one if each of its admissible values is consistent with some admissible value of the second variable. Formally, a variable \( x_i \) is arc-consistent with another variable \( x_j \) if, for every value \( a \) in the domain of \( x_i \) there exists a value \( b \) in the domain of \( x_j \) such that \((a, b)\) satisfies the binary constraint between \( x_i \) and \( x_j \). A problem is arc consistent if every variable is arc consistent with any other one.
Unit propagation

The procedure is based on unit clauses, i.e. clauses that are composed of a single literal. If a set of clauses contains the unit clause $l$, the other clauses are simplified by the application of the two following rules:

1. every clause containing $l$ is removed
2. in every clause that contains $\neg l$ this literal is deleted.