Effective Bounding Techniques For Solving Unate and Binate Covering Problems

Xiao Yu Li
Seattle, WA, USA

Matthias F. Stallmann
Raleigh, NC, USA

Franc Brglez
Raleigh, NC, USA


Related Work

- **Unate (Set) Cover Problem (UCP)**
  - 1987: espresso, Rudell et al
  - 1993: espresso-signature, McGeer et al
  - …

- **Binate Cover Problem (BCP)**
  - 1996: scherzo, Coudert
  - 2002: bsolo, Manquinho et al
  - …
Native MinCostSat Problems

Outline

1. Background
2. Our UCP/BCP Solver
3. Experimental Results
4. Conclusions
Unate (Set) Covering

Three backup positions to fill.
Five players to choose from.
Want to minimize the number of players.

<table>
<thead>
<tr>
<th></th>
<th>Duncan</th>
<th>Francis</th>
<th>Malone</th>
<th>Stockton</th>
<th>Yao</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guard</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forward</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Center</td>
<td>1</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Unate Covering as CNF Sat

• Conjunctive Normal Form

Guard      | F ∨ S
Forward    | D ∨ M
Center     | D ∨ M ∨ Y

• Goal: minimize F + S + D + M + Y

Min-cost solutions: F = 1, D = 1, S = M = Y = 0
M = 1, S = 1, D = F = Y = 0
Binate Covering Problem

Additional constraint 1:
Malone and Stockton
are together (M ⇔ S)

\[
\overline{M} \lor S \\
M \lor \overline{S}
\]

Additional constraint 2:
Duncan and Stockton
can’t be signed together (D ∧ S)

\[
\overline{D} \lor \overline{S}
\]

Binate Covering Problem (cont.)

<table>
<thead>
<tr>
<th>constraints</th>
<th>covering matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>F \lor S</td>
<td>D</td>
</tr>
<tr>
<td>D \lor M</td>
<td>1</td>
</tr>
<tr>
<td>D \lor M \lor Y</td>
<td>1</td>
</tr>
<tr>
<td>\overline{M} \lor S</td>
<td></td>
</tr>
<tr>
<td>\overline{M} \lor \overline{S}</td>
<td></td>
</tr>
<tr>
<td>\overline{D} \lor \overline{S}</td>
<td></td>
</tr>
</tbody>
</table>
Solve MinCostSat as ILP

MinCostSat is a special case of 0-1 integer linear programming

For each constraint in the MinCostSat instance...

\[ \overline{x_1} \lor x_2 \lor \overline{x_3} \lor x_4 \]

...replace \( \overline{x_i} \) with \((1 - x_i)\):

\[ (1 - x_1) + x_2 + (1 - x_3) + x_4 \geq 1 \]

\[ -x_1 + x_2 - x_3 + x_4 \geq -1 \]
MinCostSat as ILP: an example

\[ \begin{align*}
\overline{x_1} \lor x_2 \lor \overline{x_3} \lor x_4 & \quad \rightarrow -x_1 + x_2 - x_3 + x_4 \geq -1 \\
\overline{x_1} \lor x_2 \lor x_4 & \quad \rightarrow -x_1 - x_2 + x_4 \geq -2 \\
x_1 \lor x_2 \lor x_3 & \quad \rightarrow x_1 + x_2 + x_3 \geq 1
\end{align*} \]

Assume unit cost, minimize: \( x_1 + x_2 + x_3 + x_4 \)

Technology Mapping

<table>
<thead>
<tr>
<th>Library L</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND2</td>
<td>4</td>
</tr>
<tr>
<td>OR2</td>
<td>4</td>
</tr>
<tr>
<td>ANDOR2</td>
<td>5</td>
</tr>
</tbody>
</table>

Cover A \( m_1 \lor m_4 \lor m_5 \)
Cover B \( m_2 \lor m_4 \)
Cover C \( m_3 \lor m_5 \)
\( m_2 \rightarrow m_1 \) \( \overline{m_2} \lor m_1 \)
\( m_3 \rightarrow m_1 \) \( \overline{m_3} \lor m_1 \)

vertices of \( N = \{A,B,C\} \)
matches = \{m_1,m_2,m_3,m_4,m_5\}
Our UCP/BCP Solver

Classical branch and bound

1. Upper/lower bound calculation (for a unate example)
2. Branching (variable assignment)
3. One iteration of the main algorithm
4. Effect of better (global) upper bounds and (local) lower bounds.

An Example (Steiner 9)

UB = 6
find “cover” to get UB

LB = 3
based on max. ind. set (MIS)
After assignment of a variable

![Diagram showing assignment of a variable with bounds]

UB = 4 (+ 1 = 5)

LB = 2 (+ 1 = 3)

The algorithm: one iteration

1. dequeue “best” node in priority queue
2. select a variable
   - x = 0
   - x = 1
3. create two new nodes
   - UB = 5, LB = 4
   - UB = 5, LB = 3
4. do reduction and compute bounds
5. enqueue both new nodes
Lower and Upper Bounds

Branch-and-bound solver: eclipse

The algorithm

Initialize root; Q.enqueue(root); globalUB=UB(root)

while ( Q is not empty ) {
    node = Q.dequeue();
    if ( LB(node) < globalUB ) {
        select branching variable x_i
        for ( b = 0 to 1 ) {
            n_b = (node for x_i = b)
            do reduction and bounds for n_b
            if ( UB(n_b) < globalUB ) { globalUB = UB(n_b) }
            if ( LB(n_b) < globalUB ) { Q.enqueue( n_b) }
        }
    }
}

17
Eclipse Performance Factors

• Seven Performance Factors
  1. Lower Bounding (ILP techniques)
  2. Upper Bounding (stochastic search)
  3. Search-Tree Exploration Strategies
  4. Branching Variable Selection
  5. Search Pruning
  6. Reductions
  7. Data Structures

Factor 1: Lower Bounding

Three lower-bounding strategies:
• Maximum Independent Set (MIS)
• Linear Programming Relaxation (LPR)
• Cutting Planes (CP)
Lower Bounding: MIS

- MIS - maximum independent set of rows

\[
\begin{array}{cccccc}
  & x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\
\hline
\text{row1} & 1 & 1 & & & & \\
\text{row2} & & & 1 & 1 & 1 & \\
\text{row3} & & 1 & 1 & & & \\
\text{row4} & & & & 1 & 1 & \\
\text{row5} & & & & -1 & -1 & 1 \\
\end{array}
\]

- Row 1, 2 form an independent set \( \Rightarrow \) LB = 2
- Row 1, 3, 4 form an independent set \( \Rightarrow \) LB = 3

Lower Bounding: LP Relaxation

Relax integer constraints (LPR):

- LP cost = 1.0
- Integer optimum = 2
- LP optimum is a lower bound on integer optimum
Lower Bounding: Cutting Planes (CP)

Feasible region

(0.5, 0.5)

(0.7, 1.1)

Cut

CP cuts off fractional solution but not integer solutions

Lower Bound Methods (rot.b)

(MIS2 is MIS with extra tricks)

Most effective by far: CP
**Lower Bound Methods (apex4.a)**

Most effective: CP

**Lower Bound Methods (c1908_F)**

no bounding method appears effective here …
(not a case of covering problem instance)
Next Performance Factor:

1. Lower Bounding (ILP techniques)
2. **Upper Bounding (stochastic search)**
3. Search-Tree Exploration Strategies
4. Branching Variable Selection
5. Search Pruning
6. Reductions
7. Data Structures

---

**Factor 2: Upper Bounding**

- **Stochastic local search for optimal solution:**
  1. Apply at the *root* only or *at each node*
  2. Initialization – solution based on linear programming for lower bound
- For comparison purposes we initialize the global UB to the cost of the optimal solution (oracle)
Upper Bound Methods (max1024)

we report average cpu-seconds to solve the instance with different UB strategies using the same LB (the best possible, i.e. CP)

---

Local Search Initialization

… random initialization of local search is not effective, see the table of runtimes (in cpu-seconds):

<table>
<thead>
<tr>
<th>benchmark</th>
<th>random</th>
<th>rounded LP solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex5</td>
<td>16.3</td>
<td>15.1</td>
</tr>
<tr>
<td>exam.pi</td>
<td>20.8</td>
<td>5.4</td>
</tr>
<tr>
<td>max1024</td>
<td>143.9</td>
<td>27.5</td>
</tr>
<tr>
<td>bench1.pi</td>
<td>300**</td>
<td>4.7</td>
</tr>
</tbody>
</table>

**timeout value
Factor 3: Tree-Exploration Strategy

... search using priority queue performs better than depth-first search (recursion - the usual method): (runtimes in cpu-seconds)

<table>
<thead>
<tr>
<th>benchmark</th>
<th>depth-first</th>
<th>priority*</th>
</tr>
</thead>
<tbody>
<tr>
<td>exam.pi</td>
<td>7.0</td>
<td>5.4</td>
</tr>
<tr>
<td>bench1.pi</td>
<td>7.9</td>
<td>4.7</td>
</tr>
<tr>
<td>ex5</td>
<td>16.2</td>
<td>15.1</td>
</tr>
<tr>
<td>max1024</td>
<td>74.1</td>
<td>27.5</td>
</tr>
</tbody>
</table>

* smallest lower bound first (to try to reduce UB quickly)

Experimental Results

- The solvers
  1. Scherzo [Coudert, DAC1996]
  2. Cplex (State of the art IP/ILP Solver)
  3. Eclipse-lpr (LP Relaxation)
  4. Eclipse-cp (Cutting Planes)
- The benchmarks
  – Logic minimization (unate covering)
  – FSM minimization (binate covering)
Unate Results (nodes visited)

Typical Results: Unate (runtime)
Typical Results: small binate

<table>
<thead>
<tr>
<th>number of nodes</th>
<th>f51m.b</th>
<th>jac3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Small bin. benchmarks (runtime)

<table>
<thead>
<tr>
<th>CPU seconds</th>
<th>f51m.b</th>
<th>jac3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Eclipse-cp spends a lot of time with cuts at each node (cplex visits more nodes and does cuts less frequently)

Large binate benchmarks

<table>
<thead>
<tr>
<th>nodes visited</th>
<th>apex4.a</th>
<th>rot.b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Large bin. benchmarks (nodes)

Large bin. benchmarks (runtime)

<table>
<thead>
<tr>
<th>CPU seconds</th>
<th>apex4.a</th>
<th>rot.b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cplex only does cuts at some nodes.
We visit fewer nodes...

... but spend more time
We do cuts everywhere -- could be more judicious
**latte = espresso − mincov + eclipse-cp**

<table>
<thead>
<tr>
<th>benchmark</th>
<th>espresso (hours)</th>
<th>latte (secs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>cover</td>
<td>total</td>
</tr>
<tr>
<td>prom2</td>
<td>---</td>
<td>12**</td>
</tr>
<tr>
<td>ex5</td>
<td>---</td>
<td>12**</td>
</tr>
<tr>
<td>max1024</td>
<td>---</td>
<td>12**</td>
</tr>
</tbody>
</table>

**timeout**

<table>
<thead>
<tr>
<th>benchmark</th>
<th>prod-orig</th>
<th>prod-heur</th>
<th>prod-latte</th>
</tr>
</thead>
<tbody>
<tr>
<td>prom2</td>
<td>287</td>
<td>287</td>
<td>287</td>
</tr>
<tr>
<td>ex5</td>
<td>256</td>
<td>74</td>
<td>65</td>
</tr>
<tr>
<td>max1024</td>
<td>1024</td>
<td>274</td>
<td>259</td>
</tr>
</tbody>
</table>

---

The Last Words …

**Acknowledgments:**

- for the benchmarks
- for the solvers (*aura, bsolo, espresso, scherzo*)
- for constructive reviews

**Under construction:**

- code tune-ups before the release of *eclipse*
- additional experiments
  -- for the journal submission
  -- for comprehensive web-posting of results