On-the-fly cardinality detection

Jan Elffers

KTH Royal Institute of Technology

July 8, 2019

Joint work with Jakob Nordström
The Boolean satisfiability (SAT) problem

Can variables $x_1, \ldots, x_n$ be assigned true/false to satisfy clauses $C_1, \ldots, C_m$?

$$(x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_2 \lor x_3) \land (\overline{x}_1 \lor \overline{x}_2 \lor \overline{x}_3)$$

($\overline{x}_i$ denotes negation of $x_i$)

- Many problems can be encoded as SAT: planning and scheduling, hardware and software verification, combinatorial problems.
- Dramatic progress on conflict-driven clause learning (CDCL) solvers in last 2 decades [MS96, BS97, MMZ+01].
- Exist simple problems, e.g. involving counting, on which CDCL solvers fail.
The pseudo-Boolean satisfiability (PB SAT) problem

- Pseudo-Boolean (PB) linear constraints are stronger than clauses
  Compare

\[ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 5 \]

and

\[ (x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_4) \land (x_1 \lor x_5) \land (x_1 \lor x_6) \]
\[ \land (x_2 \lor x_3) \land (x_2 \lor x_4) \land (x_2 \lor x_5) \land (x_2 \lor x_6) \]
\[ \land (x_3 \lor x_4) \land (x_3 \lor x_5) \land (x_3 \lor x_6) \]
\[ \land (x_4 \lor x_5) \land (x_4 \lor x_6) \]
\[ \land (x_5 \lor x_6) \]

- And PB reasoning exponentially more powerful in theory
- But PB solvers fail on CNFs: no stronger than CDCL
Our contribution

Extend our PB solver *RoundingSat* with *cardinality detection*.

1. Extend short clauses to cardinality constraints. For example, if all these clauses are present

\[(x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_4) \land (x_1 \lor x_5) \land (x_1 \lor x_6) \land (x_2 \lor x_3) \land (x_2 \lor x_4) \land (x_2 \lor x_5) \land (x_2 \lor x_6) \land (x_3 \lor x_4) \land (x_3 \lor x_5) \land (x_3 \lor x_6) \land (x_4 \lor x_5) \land (x_4 \lor x_6) \land (x_5 \lor x_6)\]

then

\[x_1 \lor x_2\]

can be extended to

\[x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 5\]

2. Generate new clauses to be used in cardinality detection.
Our contribution

Extend our PB solver *RoundingSat* with *cardinality detection*.

1. Extend short clauses to cardinality constraints.
Our contribution

Extend our PB solver *RoundingSat* with *cardinality detection*.

1. Extend short clauses to cardinality constraints. For example, if all these clauses are present

\[
\begin{align*}
(x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_4) \land (x_1 \lor x_5) \land (x_1 \lor x_6) \\
\land (x_2 \lor x_3) \land (x_2 \lor x_4) \land (x_2 \lor x_5) \land (x_2 \lor x_6) \\
\land (x_3 \lor x_4) \land (x_3 \lor x_5) \land (x_3 \lor x_6) \\
\land (x_4 \lor x_5) \land (x_4 \lor x_6) \\
\land (x_5 \lor x_6)
\end{align*}
\]
Our contribution

Extend our PB solver *RoundingSat* with *cardinality detection*.

1. Extend short clauses to cardinality constraints. For example, if all these clauses are present

\[
(x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_4) \land (x_1 \lor x_5) \land (x_1 \lor x_6) \\
\land (x_2 \lor x_3) \land (x_2 \lor x_4) \land (x_2 \lor x_5) \land (x_2 \lor x_6) \\
\land (x_3 \lor x_4) \land (x_3 \lor x_5) \land (x_3 \lor x_6) \\
\land (x_4 \lor x_5) \land (x_4 \lor x_6) \\
\land (x_5 \lor x_6)
\]

then \( x_1 \lor x_2 \) can be extended to

\[
x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 5
\]
Our contribution

Extend our PB solver *RoundingSat* with *cardinality detection*.

1. Extend short clauses to cardinality constraints.
   For example, if all these clauses are present

   \[(x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_4) \land (x_1 \lor x_5) \land (x_1 \lor x_6) \]
   \[\land (x_2 \lor x_3) \land (x_2 \lor x_4) \land (x_2 \lor x_5) \land (x_2 \lor x_6) \]
   \[\land (x_3 \lor x_4) \land (x_3 \lor x_5) \land (x_3 \lor x_6) \]
   \[\land (x_4 \lor x_5) \land (x_4 \lor x_6) \]
   \[\land (x_5 \lor x_6) \]

   then \(x_1 \lor x_2\) can be extended to

   \[x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 5\]

2. Generate new clauses to be used in cardinality detection.
Overview

1. If all necessary short clauses present in the formula, reconstruct cardinality constraints. (standard)
2. For the general case, also find short clauses to be used as building blocks. (new)
Overview

1. If all necessary short clauses present in the formula, reconstruct cardinality constraints. (standard)

2. For the general case, also find short clauses to be used as building blocks. (new)
Reconstructing cardinality constraints

Example

\[ F = (x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_2 \lor x_3) \land (x_1 \lor x_4) \land (x_2 \lor x_4) \]

Starting from \((x_1 \lor x_2)\),
Reconstructing cardinality constraints

Example

\[ F = (x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_2 \lor x_3) \land (x_1 \lor x_4) \land (x_2 \lor x_4) \]

Starting from \((x_1 \lor x_2)\),

- Try to add \(x_3\). \((x_1 \lor x_3)\) and \((x_2 \lor x_3)\) present, so add \(x_3\) to get \(x_1 + x_2 + x_3 \geq 2\).
Reconstructing cardinality constraints

Example

\[ F = (x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_2 \lor x_3) \land (x_1 \lor x_4) \land (x_2 \lor x_4) \]

Starting from \((x_1 \lor x_2)\),

- Try to add \(x_3\). \((x_1 \lor x_3)\) and \((x_2 \lor x_3)\) present, so add \(x_3\) to get \(x_1 + x_2 + x_3 \geq 2\).
- Then, try to add \(x_4\). \((x_3 \lor x_4)\) not present, so don’t add.
Reconstructing cardinality constraints

Example

$$F = (x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_2 \lor x_3) \land (x_1 \lor x_4) \land (x_2 \lor x_4)$$

Starting from \((x_1 \lor x_2)\),

- Try to add \(x_3\). \((x_1 \lor x_3)\) and \((x_2 \lor x_3)\) present, so add \(x_3\) to get \(x_1 + x_2 + x_3 \geq 2\).
- Then, try to add \(x_4\). \((x_3 \lor x_4)\) not present, so don’t add.

Run a greedy algorithm doing this.
Overview

1. If all necessary short clauses present in the formula, reconstruct cardinality constraints. (standard)
2. For the general case, also find short clauses to be used as building blocks. (new)
Overview

1. If all necessary short clauses present in the formula, reconstruct cardinality constraints. (standard)

2. For the general case, also find short clauses to be used as building blocks. (new)
Learning new binary clauses

Clause learning in CDCL will not learn all implied binary clauses.

Example

Let \( F = (\overline{x}_1 \lor y_1) \land (\overline{x}_2 \lor \overline{y}_1) \).

Then \( x_1 \rightarrow y_1 \rightarrow \overline{x}_2 \) and \( x_2 \rightarrow \overline{y}_1 \rightarrow \overline{x}_1 \).

CDCL cannot learn \( \overline{x}_1 \lor \overline{x}_2 \), because \( x_1 \) and \( x_2 \) would have the same decision level, contradicting UIP property.
Learning new binary clauses

Clause learning in CDCL will not learn all implied binary clauses.

Example

Let \( F = (\overline{x}_1 \lor y_1) \land (\overline{x}_2 \lor \overline{y}_1) \).

Then \( x_1 \rightarrow y_1 \rightarrow \overline{x}_2 \) and \( x_2 \rightarrow \overline{y}_1 \rightarrow \overline{x}_1 \).

CDCL cannot learn \( \overline{x}_1 \lor \overline{x}_2 \), because \( x_1 \) and \( x_2 \) would have the same decision level, contradicting UIP property.

To learn those clauses, one can do

- Preprocessing: probing (semantic cardinality detection) approach in [Biere et al., 2014]
- During the search: find cuts in the implication graph of unit propagation [our work]
Probing

\[ F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_6) \]
Probing

\[ F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_6) \]

- Set \( x_1 \) to true. Run unit propagation.

\[ \bullet \quad x_1 \]
Probing

\[ F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_6) \]

- Set \( x_1 \) to true. Run unit propagation.
Probing

\[ F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_6) \]

- Set \( x_1 \) to true. Run unit propagation.

\[ x_1 \]

\[ x_2 \]

\[ x_3 \]
$F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_6)$

- Set $x_1$ to true. Run unit propagation.

![Diagram of a function with variables $x_1$, $x_2$, $x_3$, and $x_4$.]
Probing

\[ F = (\overline{x_1} \lor x_2) \land (\overline{x_1} \lor x_3) \land (\overline{x_2} \lor x_4) \land (\overline{x_3} \lor x_5) \land (\overline{x_4} \lor \overline{x_5} \lor x_6) \]

▶ Set \( x_1 \) to true. Run unit propagation.

\[ x_1 \]

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

\[ x_3 \quad x_5 \]

\[ x_2 \quad x_4 \]
Probing

\[ F = (x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_2 \lor x_4) \land (x_3 \lor x_5) \land (x_4 \lor x_5 \lor x_6) \]

- Set \( x_1 \) to true. Run unit propagation.
Probing

\[ F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor x_5 \lor x_6) \]

- Set \( x_1 \) to true. Run unit propagation.

\[ x_2, x_3, x_4, x_5 \text{ and } x_6 \text{ propagate.} \]

So learn \( (\overline{x}_1 \lor x_i) \) for \( i = 2, \ldots, 6 \).

- Repeat for all other literals (both polarities).
Finding cuts in the implication graph

Compute *all dominators* for each literal in the implication graph.

\[ F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_6) \]
Finding cuts in the implication graph

Compute *all dominators* for each literal in the implication graph.

\[ F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_6) \]

- \( x_1 \)
Finding cuts in the implication graph

Compute *all dominators* for each literal in the implication graph.

\[ F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_6) \]
Finding cuts in the implication graph

Compute *all dominators* for each literal in the implication graph.

\[
F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_6)
\]
Finding cuts in the implication graph

Compute *all dominators* for each literal in the implication graph.

\[ F = (\bar{x}_1 \lor x_2) \land (\bar{x}_1 \lor x_3) \land (\bar{x}_2 \lor x_4) \land (\bar{x}_3 \lor x_5) \land (\bar{x}_4 \lor \bar{x}_5 \lor x_6) \]
Finding cuts in the implication graph

Compute *all dominators* for each literal in the implication graph.

\[ F = (\overline{x}_1 \lor x_2) \land (\overline{x}_1 \lor x_3) \land (\overline{x}_2 \lor x_4) \land (\overline{x}_3 \lor x_5) \land (\overline{x}_4 \lor \overline{x}_5 \lor x_6) \]
Finding cuts in the implication graph

Compute *all dominators* for each literal in the implication graph.

\[ F = (\overline{x_1} \lor x_2) \land (\overline{x_1} \lor x_3) \land (\overline{x_2} \lor x_4) \land (\overline{x_3} \lor x_5) \land (\overline{x_4} \lor \overline{x_5} \lor x_6) \]

\( x_1 \) dominates all other nodes, so learn \((\overline{x_1} \lor x_i)\) for \(i = 2, \ldots, 6\).
Finding cuts in the implication graph

Compute *all dominators* for each literal in the implication graph.

\[ F = (\overline{x_1} \lor x_2) \land (\overline{x_1} \lor x_3) \land (\overline{x_2} \lor x_4) \land (\overline{x_3} \lor x_5) \land (\overline{x_4} \lor \overline{x_5} \lor x_6) \]

Suppose had decision \( y \) preceding \( x_1 \), which is part of the reason of \( x_2 \). In this case, \( x_1 \) no longer dominates \( x_2, x_4 \) and \( x_6 \).
Overall procedure

- During unit propagation, clauses are generated from cuts in the implication graph. 
  *These clauses are stored permanently in a database.*
- During conflict analysis, short clauses appearing as reasons are mapped to cardinality constraints using this database.
The limitation of probing

Suppose have clauses \((x_1 \lor x_2 \lor y) \land (x_1 \lor x_2 \lor \neg y)\).

- Probing does not discover \(x_1 \lor x_2\).
- But clause learning might lead to propagation \(\neg x_1 \rightarrow x_2\) (and \(\neg x_2 \rightarrow x_1\)), which can be discovered by our method.
Cardinality detection beyond binary clauses

- Dominators are single node cuts in the implication graph. Can extend the idea to detect small-size cuts (corresponds to short clauses). Detecting larger cuts $\rightarrow$ higher overhead.
- Non-binary clauses can also be transformed to cardinality constraints: similar to example at beginning of this talk.
Experimental evaluation

Compare our approach against the probing approach in [Biere et al., 2014] (using Sat4j + Riss).

- Sat4j is the pseudo-Boolean solver.
- Riss is the preprocessor to generate cardinality constraints.

Experiments:

- Pigeon hole principle with various encodings. [Biere et al., 2014]
- Two pigeons per hole principle with various encodings. (our proposal)
- Even colouring formula. (our proposal)
Table legend: \#solved (PAR2 score in minutes).

<table>
<thead>
<tr>
<th>Preprocessor/ Solver</th>
<th>#inst.</th>
<th>Syntactic(Riss) Sat4jCP</th>
<th>Probe(Riss) Sat4jCP</th>
<th>no RoundingSat-Card</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binomial</td>
<td>14</td>
<td>13 (36m)</td>
<td>7 (211m)</td>
<td>14 (20m)</td>
</tr>
<tr>
<td>Binary</td>
<td>14</td>
<td>2 (372m)</td>
<td>6 (241m)</td>
<td>7 (212m)</td>
</tr>
<tr>
<td>Sequential</td>
<td>14</td>
<td>14 (2m)</td>
<td>11 (91m)</td>
<td>13 (56m)</td>
</tr>
<tr>
<td>Product</td>
<td>14</td>
<td>11 (109m)</td>
<td>12 (63m)</td>
<td>7 (213m)</td>
</tr>
<tr>
<td>Commander</td>
<td>14</td>
<td>8 (181m)</td>
<td>12 (61m)</td>
<td>7 (212m)</td>
</tr>
<tr>
<td>Ladder</td>
<td>14</td>
<td>11 (101m)</td>
<td>10 (127m)</td>
<td>12 (85m)</td>
</tr>
</tbody>
</table>
Two pigeons per hole principle

Benchmark encoding that $2n - 1$ pigeons do not fit into $n - 1$ holes with capacity 2.

We use three encodings

- Sorter networks.
- BDDs.
- Adder networks.

All are generated by Minisat+. 
Two pigeons per hole principle
Comparison of approaches on pigeonhole problems

- If CNF encoding arc-consistent*, then preprocessing could work in theory.
- Otherwise, need our approach.

* arc-consistent: CNF encoding gives all unit implications that PB problem gives (before any learning).
Even colouring formula [Markström, 2006]

Unsatisfiable formula defined on undirected graphs.

Graphs are random 4-regular with a split edge.
Conclusion

We proposed on-the-fly cardinality detection.

- Reduces the number of reasoning steps if there are implied cardinality constraints.
- Can discover at-most-$k$ constraints for small $k$.
- Competitive with preprocessing methods and often better.
References I

