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 $C_1, \ldots, C_m$?

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$$

($\overline{x}_i$ 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

$$\begin{aligned}(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_5) \wedge (x_1 \vee x_6)\\ \wedge (x_2 \vee x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_6)\\ \wedge (x_3 \vee x_4) \wedge (x_3 \vee x_5) \wedge (x_3 \vee x_6)\\ \wedge (x_4 \vee x_5) \wedge (x_4 \vee x_6)\\ \wedge (x_5 \vee x_6)\end{aligned}$$

- 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*.

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

$$
\begin{aligned}
(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_5) \wedge (x_1 \vee x_6) \\
\wedge (x_2 \vee x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_6) \\
\wedge (x_3 \vee x_4) \wedge (x_3 \vee x_5) \wedge (x_3 \vee x_6) \\
\wedge (x_4 \vee x_5) \wedge (x_4 \vee x_6) \\
\wedge (x_5 \vee x_6)
\end{aligned}
$$

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

$$
\begin{aligned}
(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_5) \wedge (x_1 \vee x_6) \\
\wedge (x_2 \vee x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_6) \\
\wedge (x_3 \vee x_4) \wedge (x_3 \vee x_5) \wedge (x_3 \vee x_6) \\
\wedge (x_4 \vee x_5) \wedge (x_4 \vee x_6) \\
\wedge (x_5 \vee x_6)
\end{aligned}
$$

   then $x_1 \vee x_2$ 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

$$
\begin{aligned}
(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_1 \vee x_5) \wedge (x_1 \vee x_6) \\
\wedge (x_2 \vee x_3) \wedge (x_2 \vee x_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_6) \\
\wedge (x_3 \vee x_4) \wedge (x_3 \vee x_5) \wedge (x_3 \vee x_6) \\
\wedge (x_4 \vee x_5) \wedge (x_4 \vee x_6) \\
\wedge (x_5 \vee x_6)
\end{aligned}
$$

   then $x_1 \vee x_2$ 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

$$F = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_4)$$

Starting from $(x_1 \vee x_2)$,

- Try to add x_3. $(x_1 \vee x_3)$ and $(x_2 \vee x_3)$ present, so add x_3 to get $x_1 + x_2 + x_3 \geq 2$.

Reconstructing cardinality constraints

Example

$$F = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_4)$$

Starting from $(x_1 \vee x_2)$,

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

Reconstructing cardinality constraints

Example

$$F = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_4)$$

Starting from $(x_1 \vee x_2)$,

- Try to add x_3. $(x_1 \vee x_3)$ and $(x_2 \vee x_3)$ present, so add x_3 to get $x_1 + x_2 + x_3 \geq 2$.
- Then, try to add x_4. $(x_3 \vee x_4)$ 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 $F = (\overline{x}_1 \vee y_1) \wedge (\overline{x}_2 \vee \overline{y}_1)$.
Then $x_1 \rightarrow y_1 \rightarrow \overline{x}_2$ and $x_2 \rightarrow \overline{y}_1 \rightarrow \overline{x}_1$.
CDCL cannot learn $\overline{x}_1 \vee \overline{x}_2$, 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 $F = (\overline{x}_1 \vee y_1) \wedge (\overline{x}_2 \vee \overline{y}_1)$.
Then $x_1 \rightarrow y_1 \rightarrow \overline{x}_2$ and $x_2 \rightarrow \overline{y}_1 \rightarrow \overline{x}_1$.
CDCL cannot learn $\overline{x}_1 \vee \overline{x}_2$, 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]

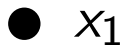
Probing

$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$

Probing

$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$

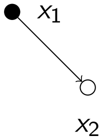
- Set x_1 to true. Run unit propagation.



Probing

$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$

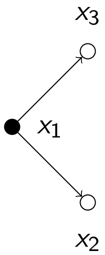
- Set x_1 to true. Run unit propagation.



Probing

$$F = (\overline{x}_1 \vee x_2) \wedge {\color{red}(\overline{x}_1 \vee x_3)} \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$

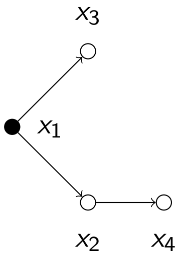
- Set x_1 to true. Run unit propagation.



Probing

$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$

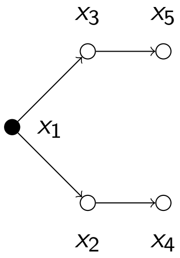
- Set x_1 to true. Run unit propagation.



Probing

$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$

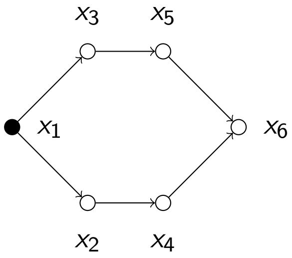
- Set x_1 to true. Run unit propagation.



Probing

$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$

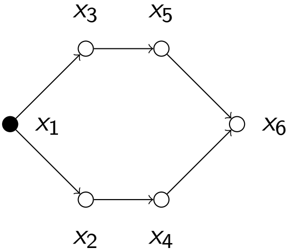
- Set x_1 to true. Run unit propagation.



Probing

$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$

- Set x_1 to true. Run unit propagation.



  x_2, x_3, x_4, x_5 and x_6 propagate.
  So learn $(\overline{x}_1 \vee x_i)$ for $i = 2, \ldots, 6$.
- Repeat for all other literals (both polarities).

Finding cuts in the implication graph

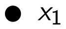
Compute *all dominators* for each literal in the implication graph.

$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$

Finding cuts in the implication graph

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

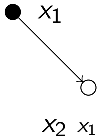
$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$



Finding cuts in the implication graph

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

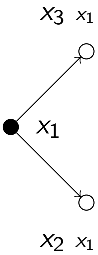
$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$



Finding cuts in the implication graph

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

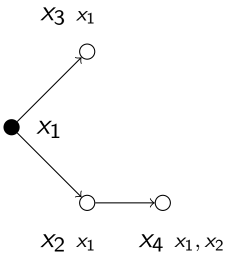
$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$



Finding cuts in the implication graph

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

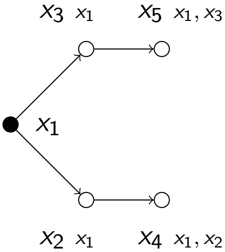
$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$



Finding cuts in the implication graph

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

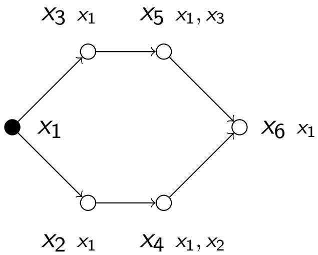
$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$



Finding cuts in the implication graph

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

$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$

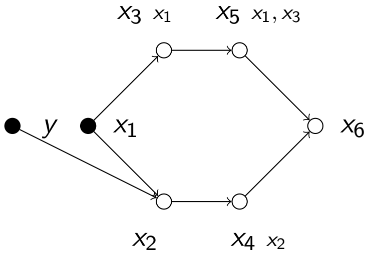


x_1 dominates all other nodes, so learn $(\overline{x}_1 \vee x_i)$ for $i = 2, \ldots, 6$.

Finding cuts in the implication graph

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

$$F = (\overline{x}_1 \vee x_2) \wedge (\overline{x}_1 \vee x_3) \wedge (\overline{x}_2 \vee x_4) \wedge (\overline{x}_3 \vee x_5) \wedge (\overline{x}_4 \vee \overline{x}_5 \vee x_6)$$



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 $(x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \overline{y})$.

- Probing does not discover $x_1 \vee x_2$.
- But clause learning might lead to propagation $\overline{x}_1 \rightarrow x_2$ (and $\overline{x}_2 \rightarrow x_1$), 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)

Pigeonhole principle

Table legend: #solved (PAR2 score in minutes).

Preprocessor Solver	#inst.	Syntactic(Riss) Sat4jCP	Probe(Riss) Sat4jCP	no RoundingSat-Card
Binomial	14	13 (36m)	7 (211m)	14 (20m)
Binary	14	2 (372m)	6 (241m)	7 (212m)
Sequential	14	14 (2m)	11 (91m)	13 (56m)
Product	14	11 (109m)	12 (63m)	7 (213m)
Commander	14	8 (181m)	12 (61m)	7 (212m)
Ladder	14	11 (101m)	10 (127m)	12 (85m)

Two pigeons per hole principle

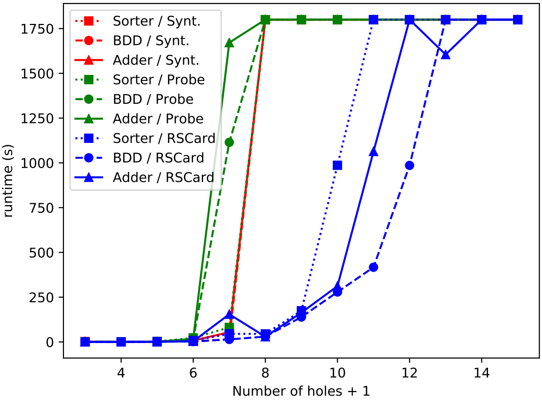
Benchmark encoding that $2n - 1$ pigeons do not fit into $n - 1$ holes with capacity 2.

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

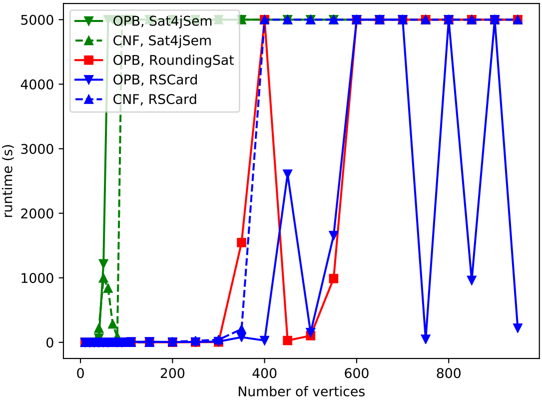
- If CNF encoding arc-consistent*, then preprocessing could work in theory.
- Otherwise, need our approach.

* 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

Roberto J. Bayardo Jr. and Robert Schrag.
Using CSP look-back techniques to solve real-world SAT instances.
In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
Chaff: Engineering an efficient SAT solver.
In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

João P. Marques-Silva and Karem A. Sakallah.
GRASP—a new search algorithm for satisfiability.
In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD '96)*, pages 220–227, November 1996.