A Cardinality Solver: more expressive constraints for free

Mark H. Liffiton and Jordyn C. Maglalang / {mliffito,jmaglala}@iwu.edu

Code: http://git.io/minicard

MiniCARD = MiniSAT + **Free, Fast** Cardinality

This work generalizes a state-of-the-art SAT solver into a “cardinality solver” we call MiniCARD. Using a watched-literal scheme for native cardinality constraints, MiniCARD greatly outperforms CNF encodings of cardinality constraints on all pure-cardinality instances tested. The modifications to the solver are minimal, and it retains its performance on pure CNF instances. With easy implementation and no cost, any CDCL SAT solver can be extended to be a cardinality solver “for free.”

Ongoing Work

- “MiniCARD+”: Solving pseudo-Boolean constraints with an encoding to cardinality.
  - Preliminary results are encouraging.
  - MiniCARD+ with a trivial encoding outperforms several dedicated PB solvers.
- Pre-processing for cardinality constraints and mixed CNF/cardinality instances.
- Investigate impact of cardinality implementation on relaxation variable var-ordering.

Cardinality Constraints

Cardinality constraints place a bound on the number of literals in a set that are assigned True. Given a set of n literals $\{a_1, a_2, \ldots, a_n\}$ and an integer bound k, s.t. $0 \leq k \leq n$, a cardinality constraint is defined as

$$\sum_{i=1}^{n} a_i \lesseqqgtr k$$

where $\lesseqqgtr$ is from $\{\leq, =, \geq\}$, forming *AtMost*, *Equals*, and *AtLeast* constraints, respectively. Note that a clause is simply an AtLeast constraint with a bound of 1:

$$(a_1 \vee a_2 \vee \ldots \vee a_n) \equiv \textstyle\sum_{i=1}^{n} a_i \geq 1$$

Common Approach: CNF

Cardinality constraints are commonly encoded into CNF. Several encodings have been proposed.

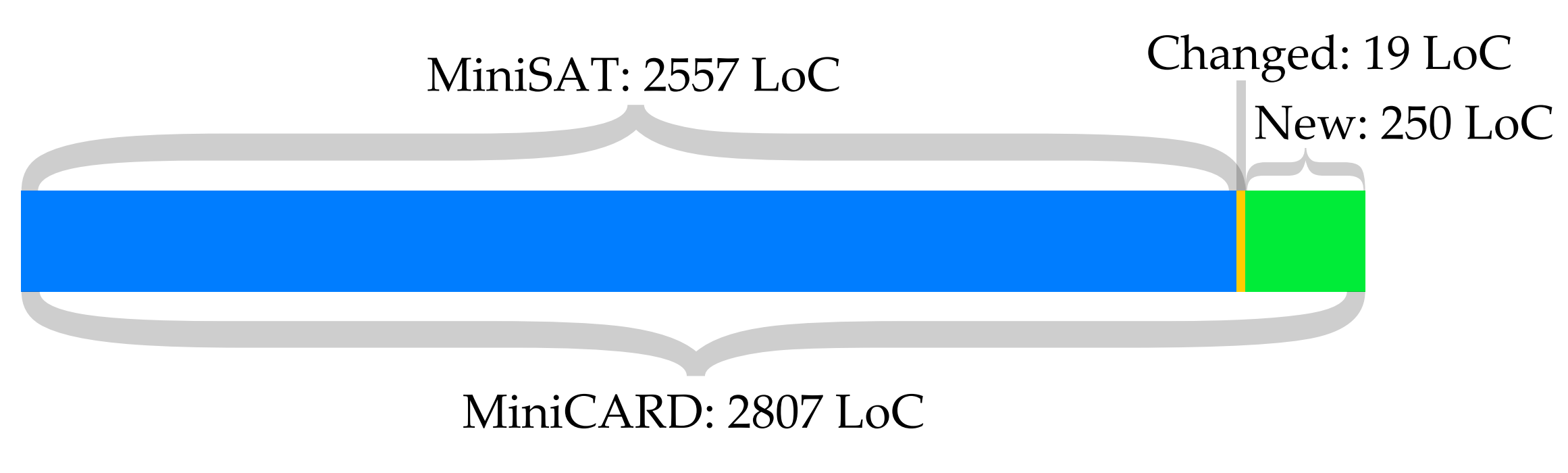
- **Binary Decision Diagrams** (BDDs) [4]: The BDD form of an AtMost constraint is translated into CNF by modeling each node of the BDD as a multiplexer.
  - $O(n \cdot k)$ clauses
  - “BDD” in the results
- Sorting Networks [4]: A network of comparators sorts the values of an AtMost’s literals; setting $n-k$ of the outputs to False enforces the bound.
- Cardinality Networks [1]: Improve on sorting networks by simplifying and using half the clauses (3 per comparator, as opposed to 6).
- **Pairwise Cardinality Networks** [2]: Improve on cardinality networks with an alternative splitting method. Can use 3-clause comparator, as well.
  - $O(n \log^2 k)$ clauses
  - “PCN” and “PCN3” (3-clause comparators) in the results

MiniCARD Implementation

We generalized the clause data structures and algorithms in MiniSAT to handle cardinality constraints directly:

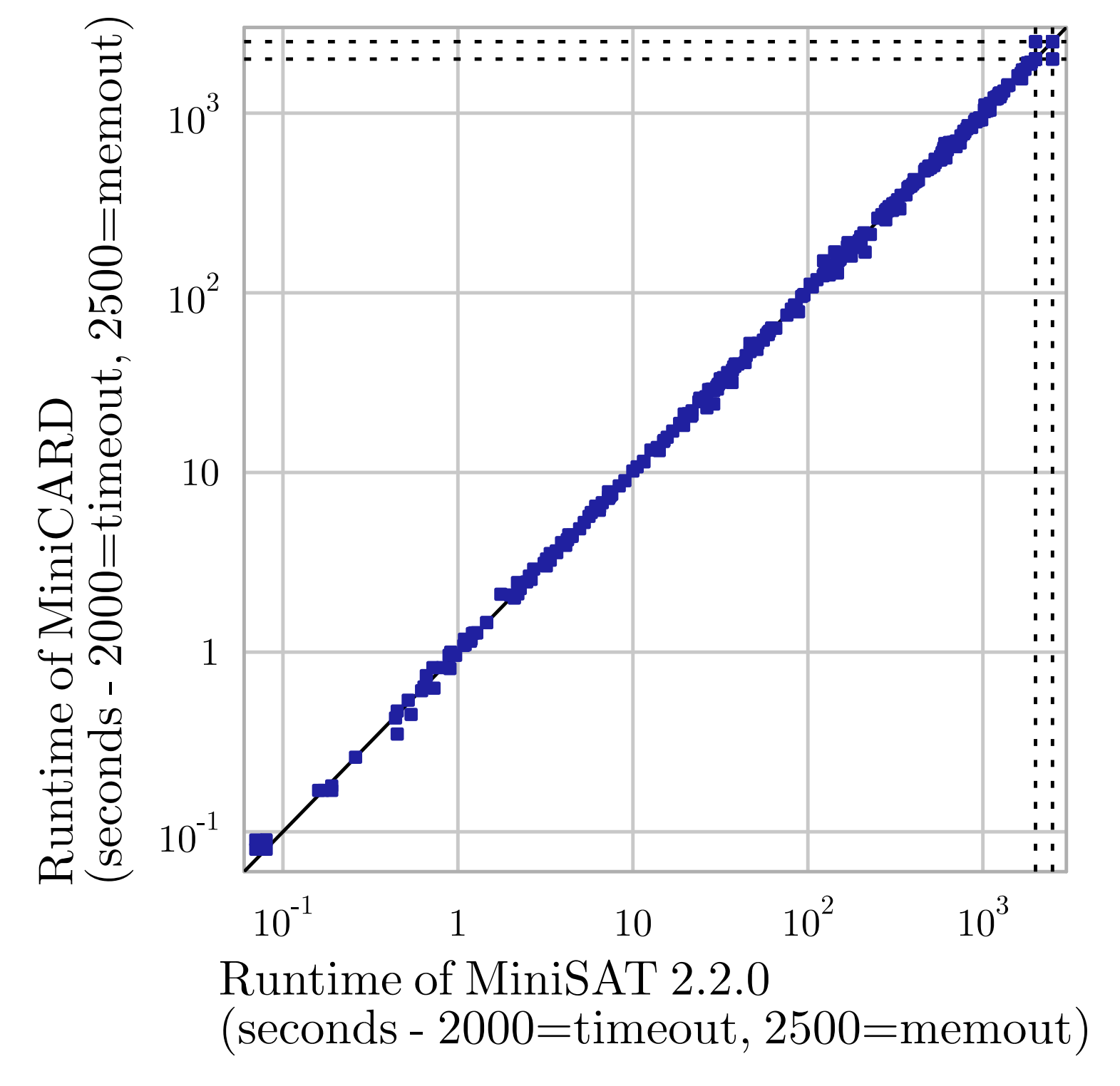
- Extended clause data structure with one-bit flag indicating cardinality/clause and an optional bound (k)
- Extended 2-watched-literals technique to m-watched-literals ($m = n - k + 1$) for cardinality constraints [3]
- Conflict detection, clause-learning, etc. unchanged

# Lines of code (non-whitespace, non-comment):



“Free”: Impact on CNF-solving

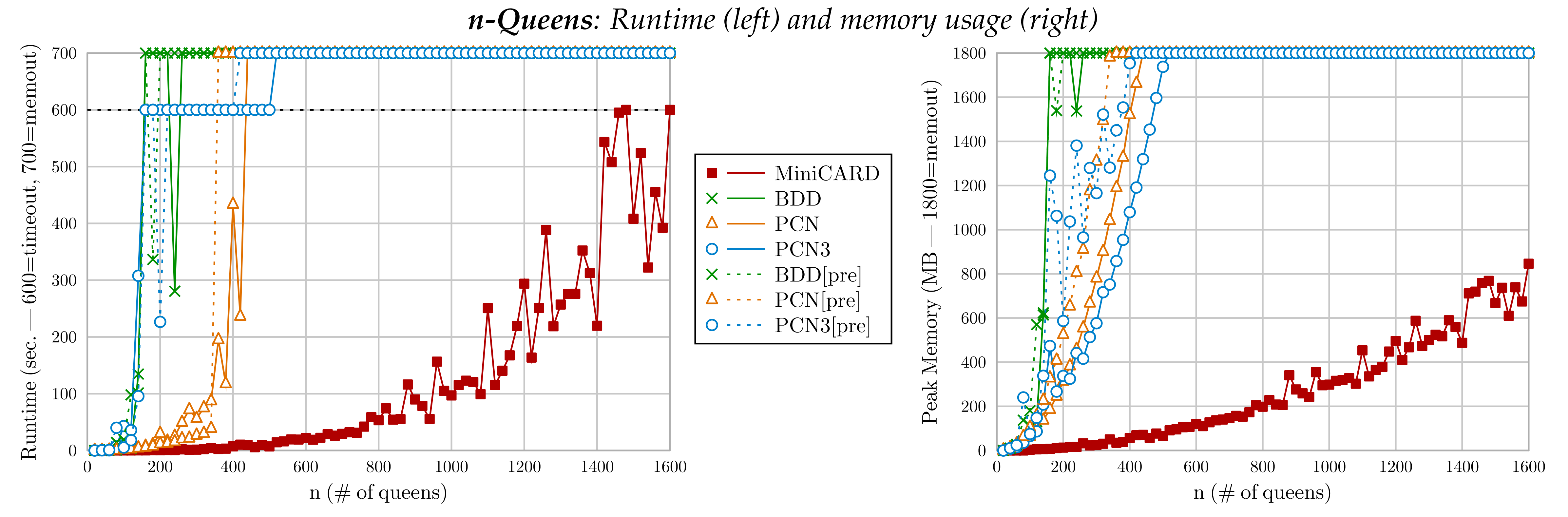
We ran both MiniSAT 2.2.0 and MiniCARD on the CNF instances from the 2011 SAT Competition. The average change in runtime from 2.2.0 to MiniCARD is 0.78%.



“Fast”: MiniCARD vs CNF Encodings: Pure Cardinality

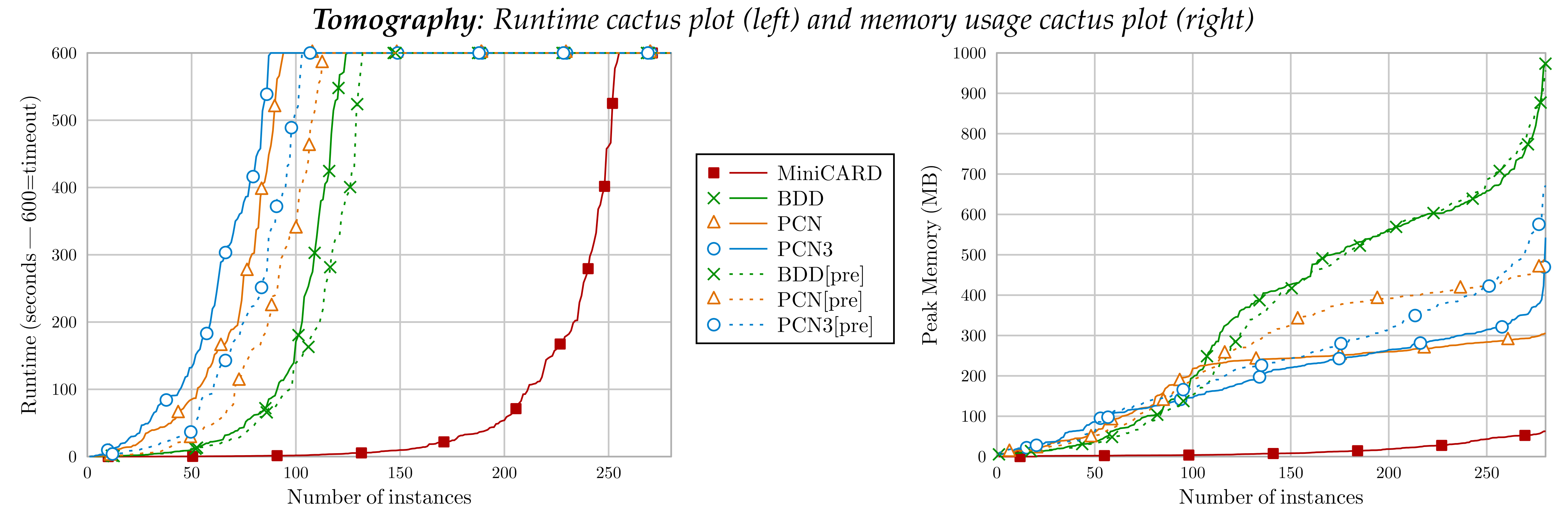
We compared MiniCARD to several several CNF encodings of cardinality constraints with instances from the n-queens problem (all AtMost 1 constraints) and the tomography problem (also on an $n \times n$ grid, generalizes n-queens with AtMost bounds ranging from 1 to n). CNF-encoded instances were solved with MiniSAT 2.2.0 both with pre-processing (“[pre]”) and without. Instances of n-Queens were generated for $n = 20, 40, \ldots, 1600$.

***n*-Queens**: *Runtime (left) and memory usage (right)*



Tomography instances were generated randomly for square grids for $n = 20$ to $n = 47$, with 10 instances of each size.

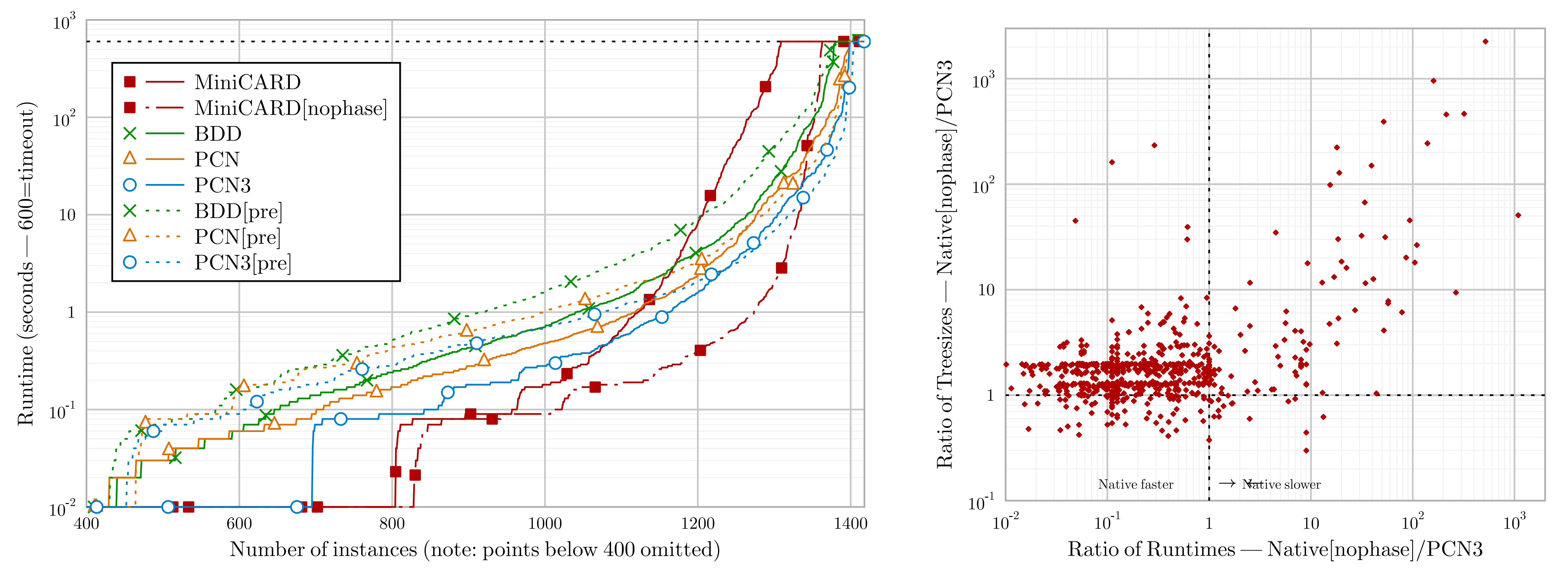
Tomography: *Runtime cactus plot (left) and memory usage cactus plot (right)*



MiniCARD vs CNF Encodings: Mixed CNF/Cardinality

Cardinality constraints are commonly applied in algorithms that use relaxation variables to analyze infeasible constraint systems. We investigated MiniCARD’s performance on MSU4, a maximum-satisfiability algorithm [5], using 1400+ MSU4 CNF+Cardinality instances from the authors of [1]. These instances have far fewer cardinality constraints than clauses, but the cardinality constraints have a large impact on the solver’s **variable ordering**.

MSU4: *Runtime cactus plot (left) and treesize/runtime comparison (right)*



MiniCARD consistently produces larger search trees, and thus longer runtimes, despite operating more efficiently than CNF encodings (in terms of decisions/sec, etc.). **Disabling phase-saving** in MiniCARD (“[nophase]”) corrects much of the poor variable ordering, but it still produces larger search trees than CNF encodings. Further work should be able to improve the variable ordering to match the search tree sizes produced by CNF encodings.

References

[1] R. Asín, R. Nieuwenhuis, A. Oliveras, and E. Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16:195–221, 2011.

[2] M. Codish and M. Zazon-Ivry. Pairwise cardinality networks. In *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 6355 of *LNCS*, pages 154–172, 2010.

[3] H. E. Dixon. *Automating Psuedo-Boolean Inference within a DPLL Framework*. PhD thesis, University of Oregon, 2004.

[4] N. Eén and N. Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:1–26, 2006.

[5] J. Marques-Silva and J. Planes. Algorithms for maximum satisfiability using unsatisfiable cores. In *Proceedings of the Conference on Design, Automation, and Test in Europe (DATE’08)*, Mar. 2008.