Cardinality Constraints in Automated Reasoning

Improving State-of-the-Art Satisfiability Solving on Hard Problems

NDSEG NATIONAL DEFENSE SCIENCE AND ENGINEERING GRADUATE FELLOWSHIP PROGRAM

Computer Science

JOSEPH E. REEVES

Carnegie Mellon University

Fellowship Year: 2022

Sponsoring Agency: AFOSR

Advisor/Mentor: Daniel J. Zelik

INTRODUCTION

Cardinality constraints appear frequently in automated reasoning (AR) problems. For example:

- "At least *k* packages need to be delivered."

$$Package_1 + Package_2 + \ldots + Package_n > k$$

- "At most *b* packages can be on a truck."

$$Package_1 + Package_2 + \ldots + Package_n < b$$

- "At most *m* trucks can be scheduled."

$$Truck_1 + Truck_2 + \ldots + Truck_n < m$$

Problem: Cardinality constraints are **not allowed as input** to satisfiability (SAT) solvers. They must be **encoded** into many simpler constraints.

Consequences: Implementing a good encoding is **error-prone**, and encodings **hide structural information** about the problem from the solver.

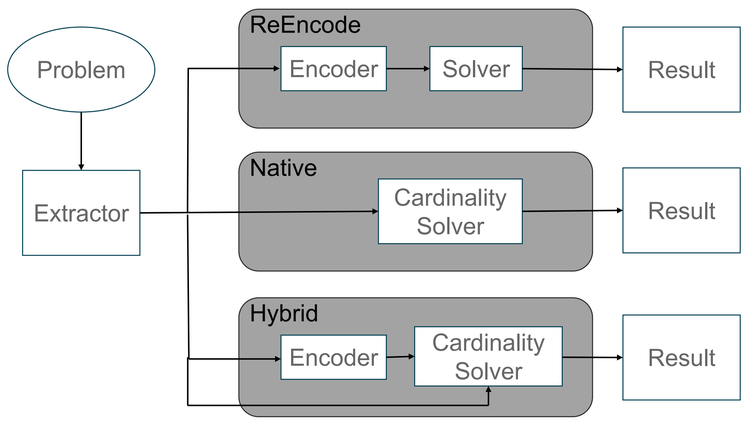
Proposed Solution: **Extend the input** of SAT solvers to include cardinality constraints, allowing solvers to handle constraints **natively** or **reencode** them.

METHODS

We sought to **extract** cardinality constraints from existing problems and **improve** solver **performance** using a cardinality-based input.

Extraction: we developed an **extraction** tool that,
- Heuristically **selects** a set of simple constraints
- **Verifies** that the simple constraints together form a cardinality constraint

Performance: modified we modified the SoTA SAT solver CaDiCaL to handle cardinality constraints natively (i.e., direct constraint propagation)

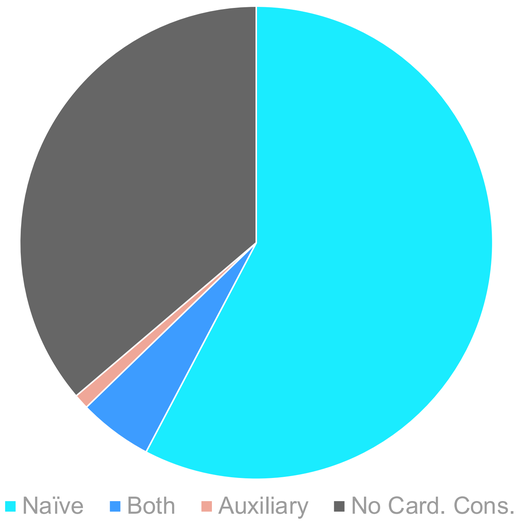


ReEncode: optimally reencode constraints
Native: natively handle cardinality constraints
Hybrid: combination of ReEncode and Native

Each solver configuration is **proof-producing**, meaning the output of the solver can be checked by a formally verified proof checker.

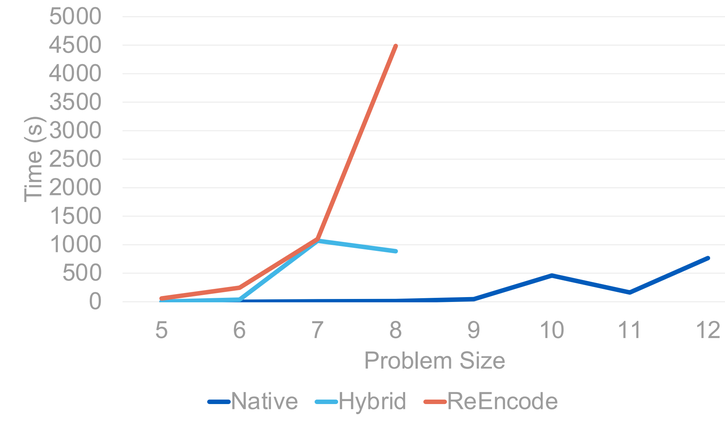
DATA ANALYSIS

1. Extracting cardinality constraints from SAT Competition Anniversary Track benchmarks.



- Cardinality constraints are **everywhere** (in over 2/3 of the benchmarks)
- Frequently using **naïve** encoding, known to be **bad** for larger constraints [1]

2. Solving extracted problems.



- Our configurations perform **better** than the default
- Configurations work well on different problems (satisfiable = problem has solutions, unsatisfiable = problem does not have solutions)
- **Hybrid** approaches **middle ground** between ReEncode and Native

3. Solving hard combinatorial problems.

- **Native handling** is necessary to **scale** for some hard problems

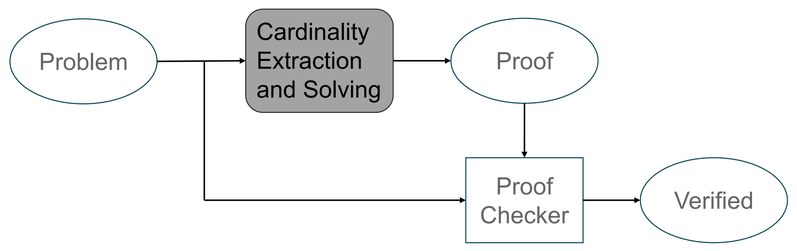
RESULTS

- Prevalence: Our extraction results show that many problems in both academia and industry contain cardinality constraints
- Extraction/Reencoding: Our extractor can find naïve encodings and reencode them with better encodings, improving performance on unsatisfiable problems
- Performance: The native cardinality handling performs well on satisfiable problems, and the hybrid approach solves many satisfiable and unsatisfiable problems
- Scaling: The native cardinality handling is effective on hard combinatorial problems

More details can be found in our accepted paper [1]

BENEFITS TO DOD

- AR is used for software and hardware verification, theorem proving, etc.
- **Verification** efforts for **safety-critical** projects are enhanced by proof-producing tools
- A **proof** is a certificate of a solver's reasoning
- Proofs can be checked by **independently verified** tools called proof-checkers [3]
- Solvers may be buggy, so a **verified proof** is necessary to establish **trust**



CONCLUSION

Our extraction tool showed cardinality constraints appear frequently in problems, yet they are not supported by most SAT solvers. Extending the input format to include cardinality constraints would both make encoding problems easier and improve solver performance. We demonstrate this through three proof-producing solving configurations that handle a cardinality-based input and outperform the default solver on several problems.

REFERENCES

[1] Magnus Bjork. Successful sat encoding techniques. Journal on Satisfiability, Boolean Modeling and Computation, 7(4):189–201, 2011.

[2] Joseph E. Reeves, Marijn J. H. Heule, and Randal E. Bryant. From clauses to klauses. In Computer Aided Verification (CAV), 2024.

[3] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Theory and Applications of Satisfiability Testing (SAT), volume 8561 of LNCS, pages 422–429, 2014.

Anticipated Graduation Spring 2026

Email: jereeves@andrew.cmu.edu

Web: https://www.cs.cmu.edu/~jereeves/