Problem Partitioning via Proof Prefixes

Zachary Battleman
Computer Science Department, Carnegie Mellon University

Joseph E. Reeves
Computer Science Department, Carnegie Mellon University

Marijn J. H. Heule
Computer Science Department, Carnegie Mellon University

1 Introduction

Satisfiability (SAT) solvers have proven to be invaluable tools for solving large problems of interest to both theorists and industry practitioners. Over the last decade and a half, substantial efforts have focused on parallelizing SAT, leading to the cube-and-conquer (CnC) [5] paradigm. A CnC solver partitions the input formula into numerous subproblems that can be solved independently in parallel. This strategy has successfully tackled longstanding open problems in mathematics, including the Pythagorean Triples problem [4], Schur Number Five [3], and the Empty Hexagon problem [6].

Historically, lookahead techniques proved remarkably effective at selecting splitting variables, often enabling superlinear speedups even on thousands of cores [5]. However, most of the significant successes in the last five years have depended on expert-crafted manual partitions [2, 6, 8, 9]. This prevents many potential users of CnC to solve their problems effectively. We propose two novel, automated partitioning methods to overcome these issues.

Our first cubing approach builds on the information contained in clausal proofs produced by SAT solvers. A *clausal proof* is a sequence of redundant clauses (i.e., clauses whose addition preserves satisfiability), ending with the empty clause to prove unsatisfiability. A central insight of this paper is that *prefixes* of clausal proofs can serve as effective stand-ins for complete proofs, and that the variables occurring in these proof prefixes provide a powerful heuristic for guiding partitioning decisions.

Our second cubing approach is for problems containing a set of clauses and one large cardinality constraint. Such problems appear frequently in the constraint optimization setting with the cardinality constraint representing some resource bound. In contrast to the first approach, this technique uses a semantic understanding of auxiliary variables to produce a good problem partition.

2 Partitioning Techniques

2.1 Proof Prefix Based Splitting

Given a formula φ, we run it with an off-the-shelf solver until a desired number of clauses are added to the proof emitted. Once the proof prefix is known, we count the variable occurrences in the proof, both positive and negative, and pick the most frequently occurring variable as the next variable to split on. After obtaining a splitting variable x, we create new formulas, $\varphi \wedge x$ as well as $\varphi \wedge \overline{x}$, and restart the process on both formulas. Naively, generating a complete partition in this manner, where each cube has size d, would require generating $O(2^d)$ proof prefixes, which would be prohibitively expensive. To get around this, we view this procedure as consisting of layers, and at each layer we generate a single variable by sampling proof prefixes from a constant number of formulas in the layer, combining the proofs for our variable extraction heuristic, and then extending all formulas by that variable. In other words, every cube in the partition will contain different polarities of the

Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

same set of variables. We call this a *static* partition. Notably, an effective static split can drastically improve our high-level understanding of these proofs by exposing the solver's underlying reasoning and highlighting the variables it deems most significant. This technique was developed into a tool called Proofix which can sit on top of any proof-producing CDCL solver.

2.2 Totalizer Based Splitting

We also developed a partitioning technique based on auxiliary variables from the totalizer encoding [1] - a common cardinality-constraint used in problems with resource bounds. We assume that the problem is given in the cardinality-based input at-least-k conjunctive normal form (KNF [7]), but the cardinality constraint may be encoded as an at-most-k constraint by negating the literals and modifying the bound. The totalizer is structured as a binary tree that incrementally counts the number of true data literals at each node. Data literals form the leaves, and each node has auxiliary variables representing the unary count from the sum of its children counters. The root of the tree is then a set of variables, $o_1, \ldots, o_n$ where o_k is true if at least k leaves are true.

In order to turn this cardinality-constraint into a partitioning method, we observe that the truth values of auxiliary variables in the interior of the tree designate how many true data literals are among its children and potentially its sibling nodes. Therefore, we can use a heuristic to determine which auxiliary variables correspond to balanced partitions of the constraints on the data literals. This technique was also developed into a tool.

3 Results

We evaluated Proofix against the state of the art CnC solver, March [5], on SAT competition formulas as well as difficult combinatorial problems. On the SAT competition formulas, Proofix outperformed March on 61% of problems, and in several cases was between 10 and 1,000 times faster. Moreover, comparing directly to CaDiCaL itself, using 32 cores, Proofix is able to find partitions which result in improved Wall Clock times 65% of the time, up to a 100× performance improvement. On the selected difficult combinatorial problems that we tested, Proofix outperformed March in all three of our tests – both in single-core and 32-core performance. In one instance, Proofix was able to solve a formula in 2,200 seconds on which March timed out after several of its subproblems took more than 10,000.

Moreover, we used formulas from the MaxSAT competition, as well as the combinatorial problems, to evaluate the totalizer splitting technique. It outperformed March on 4 out of the 5 difficult problems we selected from the competition, as well as 2 out of the 3 combinatorial problems. However, it is worth noting that in most cases, Proofix performed better.

4 Conclusion

In this paper we presented two novel techniques for automatically partitioning SAT formulas, one based on proof prefixes and the other based on the totalizer encoding. We demonstrated that the limitation to static partitions is not a major setback, while also providing numerous qualitative benefits towards explainability. Finally, we developed tools for both of the splitting techniques and demonstrated that the techniques perform better than the state-of-the-art tool on numerous problems. There are several questions left open for future work:

- What makes some proofs more amenable than others for splitting, and is this property identifiable in a prefix? In other words, can we detect when a prefix is unlikely to yield a good partition?
- Given that we can find good variables for a partition, is there a way to automatically turn them into a dynamic partition, or generalize them from their semantic meaning?

The details of the techniques, experiments, and results discussed in this extended abstract and can be found in the full paper in the SAT '25 proceedings.

References

1. Olivier Bailleux and Yacine Boufkhad. Efficient cnf encoding of boolean cardinality constraints. In Francesca Rossi, editor, *Principles and Practice of Constraint Programming (CP)*, pages 108–122, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
2. Joshua Brakensiek, Marijn J. H. Heule, John Mackey, and David Narváez. The resolution of keller's conjecture. In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 48–65, Cham, 2020. Springer International Publishing.
3. Marijn J. H. Heule. Schur number five. In *AAAI Conference on Artificial Intelligence*, 2018.
4. Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer. In *Theory and Applications of Satisfiability Testing (SAT)*, volume 9710 of *LNCS*, pages 228–245. Springer, 2016.
5. Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding cdcl sat solvers by lookaheads. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *Hardware and Software: Verification and Testing*, pages 50–65, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
6. Marijn J. H. Heule and Manfred Scheucher. Happy ending: An empty hexagon in every set of 30 points. In Bernd Finkbeiner and Laura Kovács, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 61–80, Cham, 2024. Springer Nature Switzerland.
7. Joseph E. Reeves, Marijn J. H. Heule, and Randal E. Bryant. From Clauses to Klauses. In Arie Gurfinkel and Vijay Ganesh, editors, *Computer Aided Verification (CAV)*, volume 14681 of *Lecture Notes in Computer Science*, pages 110–132. Springer, 2024.
8. Bernardo Subercaseaux and Marijn Heule. Toward optimal radio colorings of hypercubes via sat-solving. In Ruzica Piskac and Andrei Voronkov, editors, *Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 94 of *EPiC Series in Computing*, pages 386–404. EasyChair, 2023.
9. Bernardo Subercaseaux, John Mackey, Marijn J. H. Heule, and Ruben Martins. Automated mathematical discovery and verification: Minimizing pentagons in the plane. In Andrea Kohlhase and Laura Kovács, editors, *Intelligent Computer Mathematics*, pages 21–41, Cham, 2024. Springer Nature Switzerland.