Partitioning Methods for Satisfiability Testing on Large Formulas (Extended Abstract)

Tai Joon Park Allen Van Gelder

Computer Science Dept., 225 AS
University of California
Santa Cruz, CA 95064 U.S.A.
E-mail: {tjpark,avg}@cse.ucsc.edu

Abstract. Methods for partitioning large propositional formulas are investigated, with the goal of producing a set of smaller formulas whose satisfiability can be determined within reasonable time frames by known algorithms. CNF formula partitioning can be viewed as hypergraph partitioning, which has been studied extensively in VLSI design. Although CNF formulas have been considered as hypergraphs before, we found that this viewpoint was not productive for partitioning, and we introduce a new viewpoint in the dual hypergraph. Hypergraph partitioning technology from VLSI design is adapted to this problem. The overall goal of satisfiability testing requires criteria different from those used in VLSI design. Several heuristics are described, and investigated experimentally. Some formulas from circuit applications that were extremely difficult or impossible for existing algorithms have been solved. However, the method is not useful on formulas with little or no "structure", such as randomly generated formulas.

1 Introduction

The propositional satisfiability decision problem arises frequently as a subproblem in other applications, such as automated verification and automated theorem proving. Such applications may generate very large formulas, some of which are beyond the capabilities of known algorithms. Typically, these applications incorporate a satisfiability tester, as a subroutine, that performs well for most of the formulas generated by the application. However, for some formulas it keeps on running beyond acceptable time limits. What can the application do? Some applications can afford to "give up" and try something else. In other cases, failure to solve this formula is critical, and the whole application fails. Our research is directed toward providing a "satisfiability tester of last resort", to be brought in on critical formulas where standard methods have failed.

This paper summarizes results presented at CADE-13 [PVG96]. The main idea is to partition a large difficult formula into smaller formulas that (in the worst case) must each be solved. However, due to the exponential behavior of all known satisfiability decision algorithms, the smaller formulas may be many orders of magnitude easier for the standard satisfiability subroutine. Because of the overhead of formula partitioning, this method would only be invoked when the standard subroutine was unable to solve a problem within reasonable resource limits.

The two partitioning methods (see Section 3) incorporate an existing satisfiability tester as a subroutine. The first heuristic can be combined with any complete satisfiability algorithm. However, the second heuristic requires limited interaction with the underlying satisfiability algorithm, and can be combined with most model-searching algorithms, such as variants of the Davis-Putnam-Loveland-Logemann (DPLL) scheme [DP60, DLL62]. Our study combined with an existing tester program showed greatly increased efficiency on several circuit formulas that were extremely difficult or impossible for other known methods (see Figure 3).

Both heuristics are based on partitioning the input formula into two or more subformulas. Partitioning an input formula naturally fits into the hypergraph cut problem, and it represents a process that analyzes the input formula structure. Methods from VLSI design have been adapted to this problem effectively. To be useful, the cut must achieve some degree of balance in the resulting connected components, and must be small in some sense. Except for the hyperedges that occur in multiple subformulas, the structural analysis of the input formula results in subformulas that are independent of each other.

CNF formulas have been studied as hypergraphs before [GU89, GLP93]. The normal approach is to define each clause as a hyperedge connecting all the variables, or perhaps the literals, that occur in the clause. From this viewpoint the hypergraph cut problem consists of finding a favorable set of "cut" clauses, such that, if these clauses are removed from the formula, the remaining variables (the vertices of the hypergraph) fall into two or more groups (connected components) that are not related by any remaining clause. This natural method has not proven successful on large formulas, for reasons discussed in Section 2.

The approach introduced here considers the *dual* of the above hypergraph, which is also a hypergraph. In this new viewpoint, each *variable* is defined as a hyperedge connecting all the *clauses* in which it occurs. Each clause is a vertex now. In this context the hypergraph cut problem consists of finding a favorable set of "cut" variables, such that, if these variables are removed from the formula, the remaining clauses fall into two or more groups (connected components) that are not related by any remaining variable.

2 CNF Formula Partitioning

The reason to prefer the *dual* view of hypergraph over the normal approach lies in the eventual application. At a high level, the partition is used as follows: For each partial assignment "required" by the cut set, apply the assignment to the induced subformulas F_1 and F_2, making them independent. Now try to find models of F_1 and F_2 independently. A model in this context is a partial truth assignment that satisfies the formula. If this process ever succeeds, a model for the entire formula has been found. However, to demonstrate unsatisfiability it is necessary to show that the process fails for all "required" partial assignments.

The difference between the two hypergraph views lies in what partial assignments are "required". For the usual view, the cut set is a set of clauses, and *all* partial assignments that satisfy this set of clauses are "required". The number of variables in this cut set can be significantly larger than the number of clauses, and the number of satisfying partial assignments can be exponential in the number of variables involved. The number of "required" partial assignments is not directly related to the cardinality of the cut set.

For the new view, the cut set is a set of variables. The "required" partial assignments are all partial assignments to these variables that satisfy the clauses (if any) that consist entirely of variables (positive or negative) in the cut set. While this number is exponential also, it is directly related to the cardinality of the cut set, so an algorithm to find small cut sets is more likely to achieve a useful partition.

As further motivation for the new hypergraph view, consider that a formula typically has more clauses than variables. In VLSI design, there are many more gates, which correspond to vertices, than wires, which correspond to hyperedges. Thus we expected that partitioning algorithms from that domain would transfer more effectively for the new hypergraph view.

Given an input formula F, all the variables in F are grouped into the following three classes: V_c, V_1, and V_2. The resulting classification of variables must guarantee that there exists no clause that contains both V_1 and V_2 variables.

Example 1. The input formula is

$$F = \{\overbrace{(v_1, v_2)}^{C_1}, \overbrace{(v_1, v_4)}^{C_2}, \overbrace{(-v_1, v_2, v_4)}^{C_3}, \overbrace{(-v_1, v_3)}^{C_4}, \overbrace{(v_1, -v_3)}^{C_5}\}$$

The derived *dual* hypergraph from F is shown in Figure 1. One possible partition of F is

$$F_1 = \{\overbrace{(v_1, v_2)}^{C_1}, \overbrace{(v_1, v_4)}^{C_2}, \overbrace{(-v_1, v_2, v_4)}^{C_3}\}$$
$$F_2 = \{\overbrace{(-v_1, v_3)}^{C_4}, \overbrace{(v_1, -v_3)}^{C_5}\}$$

The resulting status of the variables are $V_1 = \{v_2, v_4\}$, $V_c = \{v_1\}$, and $V_2 = \{v_3\}$. $\square$

The partition of F into F_1 and F_2 can be viewed as hypergraph cut problem, and it has been studied extensively in VLSI/PCB CAD. Among the many available hypergraph partitioning algorithms (see [PVG96] for detail), we implemented the hypergraph min-cut algorithm by Fiduccia and Mattheyses [FM82].

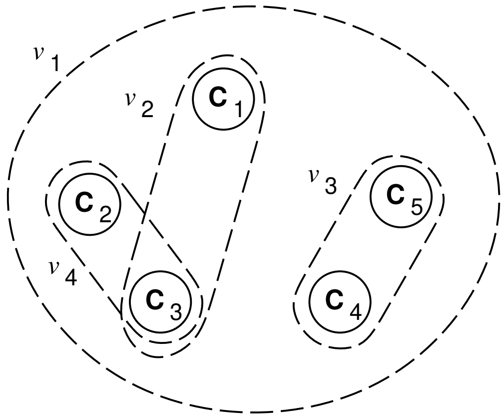

Fig. 1. The pictorial representation of the derived hypergraph.

---

3 The Two Partitioning Heuristics

The motivation behind the two partitioning heuristics is presented in Section 3.1, and we briefly discuss how the two heuristics explore the search space imposed by the cut set in Section 3.2. For further details, see [PVG96].

3.1 Motivation

Assume that a SAT tester S typically determines satisfiability of a formula F after $T(N)$ running time, where F has N variables. Assume that $T(N)$ is given by

$$T(N) = A\, 2^{\alpha N}$$

for some constants A and α. For convenience, the time unit is chosen to make $A = 1$. The α value indicates hardness of the formula class.

Both heuristics require an input formula F to be partitioned into F_1 and F_2. Let N_1 and N_2 be the number of variables in F_1 and F_2 respectively, and N_C be the size of the cut set. Then, the expected running time of the two heuristics with S is following:

$$T'(N) = 2^{N_C}(2^{\alpha N_1} + 2^{\alpha N_2})$$

Assuming that the size of the two subformulas is balanced, $T'(N)$ approaches approximately $2^{N_C}\sqrt{T(N)}$, which is much smaller than $T(N)$ when N_C is small.

3.2 Exploring The Cut Set Search Space

For each assignment to the cut variables, F_1 and F_2 simplify into formulas that have no variables in common. They can then be tested independently. F is satisfiable if and only if there is some compatible assignment to the cut variables that makes the resulting simplifications of F_1 and F_2 satisfiable.

The first heuristic exhaustively generates assignment to the cut variables until a compatible assignment is found or no more assignment can be generated. However, the second heuristic—the main innovation presented—begins by trying to satisfy one of the partitioned formulas while delaying the bindings to "cut" variables. When the formula can be satisfied with just a few cut variables bound, there is a potential to greatly reduce the search space for a compatible assignment. When the subformula F_1 is satisfiable, the model of F_1 may not have bindings to all cut variables. Then the "don't care" variables (unassigned cut variables) can have any binding when searching for a model of F_2. No matter what truth assignments are made to the "don't care" variables in F_2, those assignments cannot be conflict variables between F_1 and F_2 since there are no truth assignments made to "don't care" variables in F_1. Thus, only the binded cut variables in the model of F_1 are forced on F_2 as a set of unit clauses constraints.

4 Experimental Results

The result of partitioning test formulas is shown in Figure 2 (see [PVG96] for more detail). All of the test formulas are unsatisfiable instances. A comparison of the running time between `2cl` (a model-searching method [VGT95]) and MSAT (our partitioning method, built over `2cl`, and using the second heuristic in Section 3.2) is shown in Figure 3. In general, MSAT resulted in significant speed gain. For example, `2cl` spent about 200 CPU hours to determine the satisfiability of c5315-3, but for MSAT it took less than 5 CPU minutes. The extreme increase of efficiency for some formulas was possible because the partitioning step extracts the structural information of the input formulas, and MSAT avoids forcing unnecessary combination of truth assignment of the cut set.

5 Conclusion

We have introduced two heuristics that are based on partitioning an input formula. These two heuristics are control programs that can incorporate an existing SAT tester as a subroutine. For some of the circuit formulas, the two heuristics showed significant of gain of efficiency with little (or no) modification of the existing SAT tester. This supports our intuition that dealing with subformulas can be within the reach of existing SAT testers although the original formula may not be.

From the result shown in Figure 3, we observe that the partitioned subformulas of pret150-75 are still hard. Since the size of the cut set is only 5, we predict

formula	vars	clauses	Variables in cut	F_1	F_2
c5315-1	728	2199	6	316	406
c5315-3	728	2200	6	316	406
c2670-13	606	1642	11	218	377
c2670-16	626	1642	11	400	195
c2670-18	626	1642	10	420	176
pret150-75	150	400	5	83	62
ssa2670-127	449	1246	10	301	138

Fig. 2. Test formulas and the resulting partitions.

formula	`2cl` runtime	MSAT runtime solver	partition	total
c5315-1	> 86268	233	32	265
c5315-3	741187	233	31	264
c2670-13	20950	2214	183	2397
c2670-16	9547	1608	183	1791
c2670-18	19105	8105	188	8293
pret150-75	> 36000	14057	2	14059
ssa2670-127	1369	554	96	650

Fig. 3. Comparison of the run time between `2cl` and MSAT program. The "$>$" symbol under "`2cl` runtime" denotes that the program did not finish, and was cancelled after this amount of time. Times are CPU seconds on a Sun SPARCsystem 10/41.

that MSAT can perform better if we decompose the subformulas further. The further decomposition of these subformulas is feasible because of the small size of the cut set, and its efficiency is a future research issue.

Acknowledgments

Both of the authors were in part supported by NSF Grant CCR-9503830. We thank Tracy Larrabee and her research group for providing test formulas, and Yumi Tsuji for helping us in the process of modifying her `2cl` program.

References

[DLL62] M. Davis, G. Logemann, and D. Loveland. "A Machine Program for Theorem-Proving". *Communications of the Association for Computing Theroy*, 5:394–397, 1962.

[DP60] M. Davis and H. Putnam. "A Computing Procedure for Quantification Theory". *Journal of the Association for Computing Theroy*, 7:201–215, 1960.

[FM82] C. Fiduccia and R. Mattheyses. "A linear-time heuristic for improving network partition". In *In ACM IEEE 19th Design and Automation Conference Proceedings*, pages 175–181, June, 1982.

[GLP93] G. Gallo, G. Longo, and S. Pallottino. "Directed Hypergraph and Applications". *Discrete Applied Mathematics*, 42:177–201, 1993.

[GU89] G. Gallo and G. Urbani. "Algorithms for Testing the Satisfiability of Propositional Formulae". *Journal of Logic Programming*, 7:45–61, 1989.

[PVG96] Tai Joon Park and Allen Van Gelder. "Partitioning Methods for Satisfiability Testing on Large Formulas ". In *Proceedings 13th International Conference on Automated Deduction, New Brunswick(USA).*, volume 1104 of *Lecture Notes in Computer Science (in Artificial Intelligence)*, pages 748–762. Springer-Verlag, July/August 1996.

[VGT95] A. Van Gelder and Y.K. Tsuji. "Satisfiability Testing with More Reasoning and Less Guessing". In D. S. Johnson and M. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation and Challenge.*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1995.

This article was processed using the LaTeX macro package with LLNCS style