Partitioning Methods for Satisfiability Testing on Large Formulas

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. A propositional satisfiability tester is needed as a subroutine for many applications in automated verification, automated theorem proving, and other fields. Such applications may generate very large formulas, some of which are beyond the capabilities of known algorithms. This paper investigates methods for partitioning such formulas effectively, to produce smaller formulas within the reach of 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. 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.

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. Due to 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.

We present two partitioning heuristics for large CNF formulas. The first heuristic can be combined with any complete satisfiability algorithm. 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]. Experiments indicate that both heuristics can be very effective, but the second heuristic seems to be effective more often.

Our 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 represents a process of analyzing the structure of the input formula. 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. Heuristics based on this observation can reduce the size of the search space. 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.

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 3.

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.

1.1 Summary of Results

Our algorithm is not really another SAT tester. Rather, it employs an existing SAT tester as a part of its algorithm. When a SAT tester encounters hard formulas, the same SAT tester with our algorithm can show a significant gain of speed. Our heuristics (presented in Section 4 and Section 5) are implemented as control programs that incorporate existing SAT testers. The first heuristic can be used with any *complete* SAT tester, and the second heuristic can be used with DPLL-style, and other model-search, testers. Both heuristics rely on finding a useful hypergraph partition with a manageable number of "cut" variables.

Methods from VLSI design have been adapted to this problem effectively. The idea is to search the space of satisfying assignments to the "cut" variables to find a compatible assignment for all partitions, or to ensure that no compatible assignment exists.

The main innovation presented is the second heuristic, which 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. Intuitively, the reason is that there are many "don't cares" among the cut variables, making it more flexible to satisfy the second partitioned formula. Experimental evidence bears out this intuition.

Representation of propositional models with equivalence classes of literals permits a new branching scheme to be used with the second heuristic (Section 5.3), for the purpose of searching the assignment space of the "cut" variables efficiently. That is, the view of "assignment" is generalized to include both $r = true$ and $q = p$, where p is the "leader" of a set of equivalent literals. Viewing the assignment as a set of equations offers a different way to search the assignment space. Instead of dividing the space by $q = 1$ and $q = 0$, one can divide it by $q = p$ and $q \neq p$. Some sophisticated SAT testers employ literal equivalence detection, and are able to output a description of a model that includes such generalized assignments. The intuition here is that, if the SAT tester constrained $q = p$ in one partitioned formula, then it is more likely that $q \neq p$ will lead immediately to unsatisfiability, making it unnecessary to test $(q = true, p = false)$ and $(q = false, p = true)$ separately. Again, experimental evidence suggests that this technique is effective, but a definite conclusion is not supported.

1.2 Overview of Methodology

The main idea is to partition a CNF formula F into two subformulas, F_1 and F_2 such that they share few variables and are about the same size. Since SAT testers are exponential time, working with smaller formulas can result in a huge time reduction (see Section 4.2 for discussion). However, because of the overhead of partitioning, this method is intended to deal only with hard formulas. The partition of F into F_1 and F_2 naturally fits into a hypergraph cut problem (see Definition 2 in Section 2.1).

The variables that occur in both partitions are called *cut* variables. 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. It is also possible that F_1 or F_2 is unsatisfiable in its own right, without binding any cut variables. More sophisticated possibilities are discussed later.

1.3 Related Work in Hypergraph Partitioning

In VLSI/PCB CAD, the hypergraph cut problem has been studied extensively. The hypergraph min-cut bisection problem (see Section 2.1) is viewed as a natural abstraction of VLSI and PCB clustering placement problems [Don88]. Finding a min-cut bisection of a hypergraph is an NP-hard problem [GJ79].

One approximation method is to transform the hypergraph into a graph representation, hoping to take advantage of the existing graph partitioning algorithms. However, there is no assurance that good solutions transfer from one problem to the other. Moreover, finding the min-cut bisection of a graph is also an NP-hard problem [GJ79].

There are several classes of algorithms for finding approximate min-cut bisections of a graph [KL70, KGV83, Don88, Kah91]. "Spectral" methods have been studied recently [Kah91, CSZ94], which use eigenvalues of matrices that are derived from a graph representation. Even when the bisection criterion is relaxed to a *ratio-cut* criterion [WC89] (see Section 2.1), the problem is still an NP-hard problem (by reduction from Bounded Min-Cut Graph Partition [GJ79]).

The iterative method of Fiduccia and Mattheyses [FM82] deals with a hypergraph directly instead of a graph representation. This is the method we adapted for the implementation reported here.

2 Preliminaries

In this section, we present some basic definitions regarding hypergraphs in Section 2.1. We review the basic concepts regarding CNF formulas and review the basic behavior of DPLL tester in Section 2.2 and in Section 2.3.

2.1 Hypergraph Definitions

Definition 1. A hypergraph is a pair $H = (V, E)$ where $V = \{v_1, \ldots, v_m\}$ is the set of *vertices*, and $E = \{e_1, \ldots, e_n\}$ is the set of *hyperedges*. Each $e_i \subseteq V$. If each hyperedge has cardinality exactly 2, then H is an undirected graph.

Definition 2. A *cut* in H is a partition of V into two disjoint nonempty sets V_1 and V_2. A hyperedge e_i *crosses the cut*, and is called a *cut hyperedge*, if it contains vertices in both V_1 and V_2. The set of cut hyperedges is called the *cut set*, and is denoted as E_c.

Definition 3. A bisection of H is a cut which satisfies $|C_{V_1} - C_{V_2}| \leq 1$ where $C_{V_i} = |V_i|$. Given a weighting W of the hyperedges, the minimum bisection (or min-cut bisection) of H is the bisection with minimum weight $W(E_c) = \sum W(e_i), e_i \in E_c$.

We use $W(e_i) = 1$ for all $e_i \in E$ in this study.

Definition 4. The *minimum ratio cut problem* is that given $H = (V, E)$ and a weighting W of the hyperedges, find the partition of V into two disjoint V_1 and V_2 such that $\frac{W(E_c)}{C_{V_1} \times C_{V_2}}$ is minimized.

2.2 Notation for CNF Formulas

A positive literal is a positive occurrence of a Boolean variable, and a negative literal is the negation of a Boolean variable. The negation, or complement, of a literal q is written as $-q$, with the understanding that $- -q = q$. A *clause* of a CNF formula is a disjunction of literals. A CNF formula is a conjunction of clauses.

A partial truth assignment is a partial function that maps some variables of a formula into {true, false}; it is conveniently represented by the set of *literals* that are mapped into true. It extends to a partial function on clauses: given literal q in a partial assignment, a clause C containing q is mapped to true, and is said to be *satisfied*; a clause is mapped into false if all of its literals are mapped into false; otherwise the clause is not mapped by the partial assignment.

If all the clauses in a formula are satisfied by some partial assignment, the formula is *satisfiable*, and the partial assignment is called a *model*. A model is a *total model* if it is a model and is a total assignment. A model can be extended to a total model by arbitrarily assigning truth values to the unassigned variables. A model is conventionally represented by the set of *unit clauses* that are mapped into *true* by the partial assignment. Note that this representation is a very simple formula. In a setting where a model is required, $\{q_1, \ldots, q_k\}$ denotes the formula of unit clauses $\{(q_1), \ldots, (q_k)\}$.

Definition 5. In the context of a CNF formula, two literals p and q are said to be *equivalent to each other* if there exist two binary clauses of the following form, $(p, -q)$ and $(-p, q)$.

The detection of an equivalence relation is useful since we can substitute one literal in the equivalence relation by the other literal in the relation, reducing the number of variables in the formula.

2.3 DPLL Style Tester

A basic model-search satisfiability tester was described by Davis *et al.*, and the satisfiability tester, S, works as follows.

unit clause rule If there exists a unit clause, a clause containing only one literal, S assigns *true* to the literal.

pure literal rule If there exists a literal q such that $-q$ does not occur in the formula, S assigns *true* to q.

The unit clause rule and pure literal rule are examples of simplification rules because the formula can be made smaller without changing its satisfiability status.

splitting rule If no simplification rule applies, choose some literal q and recursively try to add q to the partial assignment, then if that fails to produce a model, recursively try to add $-q$ to the partial assignment.

After applying a rule that binds a literal q into the partial assignment, a new formula is generated in which each clause containing q is deleted, and each occurrence of literal $-q$ in clause is removed, shortening the clause. If an empty clause is created, the new formula is unsatisfiable, and the procedure backtracks. If all clauses are deleted, a model has been found, and the procedure terminates.

The tester S chooses a branching literal q based on its branching heuristics. The forward guess is the assignment of true to q. If S recursively detects a contradiction, it tries $-q$ (*alternative guess*). If the alternative guess also recursively detects a contradiction, S backtracks to the previous branching variable. If there exist no more previous branching variables, S returns *unsatisfiable*.

Branching algorithms generate an enumeration tree (ET). A branching heuristic determines which variable will have what truth assignment at each node of an ET. If the truth assignment made to a branching variable is clever, it causes a recursive application of the simplification rules, and therefore the size of an ET is reduced.

3 Partitioning F into F_1 and F_2

As mentioned in the introduction, the usual way to view a CNF formula as a hypergraph associates each clause with a hyperedge. However, we shall adopt a different view, in which each *variable* is associated with a hyperedge, which produces the hypergraph dual of the usual view. The reason to prefer the new view lies in the eventual application, after a partition has been found.

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 independently for F_1 and F_2. 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.

In Section 3.1, we illustrate the concept of partitioning.[1] We show a formulation of a hypergraph from the input formula in Section 3.2.

3.1 Illustration of the Concept of Partitioning

Given an input formula F, we want to classify all the variables in F as a member of one of the following 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

$$F = \{(v_1, v_2), (v_1, v_4), (-v_1, v_2, v_4), (-v_1, v_3), (v_1, -v_3)\}$$

One possible partition of F is

$$F_1 = \{(v_1, v_2), (v_1, v_4), (-v_1, v_2, v_4)\}$$
$$F_2 = \{(-v_1, v_3), (v_1, -v_3)\}$$

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

3.2 A Hypergraph Formulation

The partition of F into F_1 and F_2 can be viewed as hypergraph cut problem, and we derive a hypergraph from a test formula. The derivation of a hypergraph from F is as follows: a clause corresponds to a vertex of a hypergraph, and the set of all clauses where a variable, v_i, occurs is a hyperedge in our formulation. If we restrict $|m_{F_1} - m_{F_2}| \leq 1$ where m_{F_i} is the total number of clauses in F_i, the V_c variables are the bisection of our hypergraph.

Example 2. The label of each clause ID number in F is to show the transformation process.

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

[1] Before partitioning, connected component analysis of F is done first. If there exist several disconnected components, each one can be treated as an independent formula to be satisfied. Hereafter, we assume F is connected.

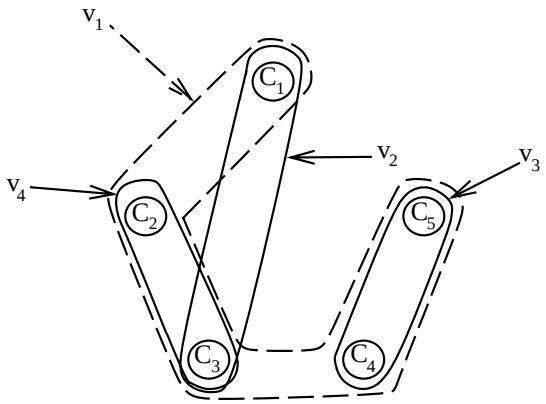

Fig. 1. The pictorial representation of hypergraph in Example 2.

The resulting hypergraph, $H = (V, E)$, is such that

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

Again, each hyperedge in E has the label of its corresponding variable in F. The pictorial representation of the resulting hypergraph is shown in Figure 1. □

Using the resulting hypergraph representation, we implemented the hypergraph min-cut algorithm by Fiduccia and Mattheyses [FM82]. The partitioning results are discussed in Section 6.

4 The First Heuristic Based On Partitioning

In Section 4.1, we present the first heuristic that can be combined with any existing complete SAT testers. Compared to the original SAT tester, the derived SAT tester can show increased efficiency in determining satisfiability if the size of cut set is usable. In Section 4.2, we discuss the reason behind the predicted increase of efficiency, and the relationship between balancing the two subformulas and the number of cut variables is investigated.

4.1 Trying All Combinations

The basic idea of the first heuristic is following. Assume that there are k cut variables as result of partitioning F into F_1 and F_2. Then there exist 2^k possible combinations of truth assignments for the k cut variables. Each combination C_i can be viewed as a set of predefined constraints of unit clauses to F_1 and F_2.

Given a combination C_i, let F_1^i be C_i combined with F_1 that is simplified, and let F_2^i be C_i combined with F_2 and simplified. If both F_1^i and F_2^i are satisfiable, the formula F is satisfiable since there exist no conflicting truth assignments between F_1^i and F_2^i. The following is the description of the first heuristic, and we call the first heuristic as `tryAll`.

1. Determine satisfiability of F_1 and F_2. If either F_1 or F_2 is unsatisfiable, return unsatisfiable and halt.
2. Generate a combination C_i. If there exists no more unique combination, return unsatisfiable and halt.
3. With the resulting combination C_i of step 2, generate F_1^i and determine satisfiability of F_1^i. If F_1^i is unsatisfiable, go to step 2. Otherwise, generate F_2^i and determine satisfiability of the F_2^i. If F_2^i is satisfiable, return satisfiable and halt. Otherwise, go to step 2.

Any complete SAT tester can be used to determine satisfiability in the step 1 & 3. Thus, `tryAll` is a control program that uses any complete tester as a subroutine.

4.2 The reason behind Expected Increase of Efficiency

Assume that a SAT tester S typically determines satisfiability of a formula F after $T(n)$ running time, where F has n variables ($n = n_{V_c} + n_{V_1} + n_{V_2}$). 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.

Assume hardness classes of F_1^i and F_2^i be β and γ respectively. Let $n_{V_1}^i$ and $n_{V_2}^i$ be the total number of variables in the subformula F_1^i and F_2^i respectively. Then, the expected running time of `tryAll` with S is following:

$$T_{tryall}(n) = 2^{n_{V_c}}(2^{\beta n_{V_1}^i} + 2^{\gamma n_{V_2}^i})$$

In the above expression, $2^{n_{V_c}}$ indicates that the `tryAll` routine examines all the possible combinations of truth assignments of V_c variables in the worst case.

The relationship between n_{V_c} and balancing the number of variables in F_1 and F_2 is derived by trying to satisfy the following inequality,[2]

$$2^{n_{V_c}}(2^{\alpha n_{V_1}^i} + 2^{\alpha n_{V_2}^i}) \leq 2^{\alpha n} \tag{1}$$

In Eq. 1, we assume that F_1^i and F_2^i are in the same hardness class as F, i.e., $\beta = \gamma = \alpha$. Let $d = |n_{V_1} - n_{V_2}|$. A sufficient condition for Eq. 1 is

$$n_{V_c} \leq \alpha \left(\frac{n-d}{2} \right) \tag{2}$$

This gives an heuristic criterion for trading off cut size and balance of the partition. In Eq. 2, as d value becomes smaller, we can have more V_c variables and still improve the predicted performance by the same amount.

[2] We assume that m_{F_i} and n_{V_i} are proportional to each other.

5 The Second Heuristic Based On Partitioning

We show the motivation behind the second heuristic in Section 5.1. The second heuristic to be combined with DPLL style testers is presented in Section 5.2. The existing DPLL style tester program needs to be modified, and the necessary modification is discussed in this section.

5.1 Motivation

When the resulting cut size is not small, the first heuristic becomes impractical to apply. However, we could greatly reduce the exponential search space of compatible assignment if the bindings to the cut variables are delayed. If 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. The following example illustrates the idea.

Example 3. The following formulas are the resulting two subformulas of F shown in Example 1.

$$F_1 = \{(v_1, v_2), (v_1, v_4), (-v_1, v_2, v_4)\}$$
$$F_2 = \{(-v_1, v_3), (v_1, -v_3)\}$$

With a model $M_{F_1} = \{v_2, v_4\}$, the subformula F_1 is satisfied. Notice that the cut variable v_1 had no chance to be assigned. With a model $M_{F_2} = \{v_1, v_3\}$ of F_2, we can build a model $M_F = \{v_1, v_2, v_3, v_4\}$ of F by combining M_{F_1} and M_{F_2}. This is possible since the cut variable v_1 is a "don't care" variable. □

The existing DPLL style tester is modified to output the partial assignment instead of a *total model* when a formula is satisfiable.

5.2 Delaying Cut Variable Binding

The following is the description of the second heuristic, and we call the second heuristic as MSAT. The idea is to try to satisfy the partitioned formulas while binding as few cut variables as possible. If there exists no satisfying assignment with the few bindings, try to search among those variables, and broaden the search only when forced to do so.

1. Determine satisfiability of F_1 and F_2. If either F_1 or F_2 is unsatisfiable, return unsatisfiable and halt. If both F_1 and F_2 are satisfiable, let M_1 be bindings made to the cut variables while satisfying F_1, and let M_2 be bindings made to the cut variables while satisfying F_2. Depending on the number of elements in M_1 and M_2, the two subformulas and their cut bindings will be renamed so that $|M_1| \leq |M_2|$.

Let $F_2^{'}$ be the combined formula of F_2 and M_1 that is simplified. Determine satisfiability of $F_2^{'}$. If it is satisfiable, return satisfiable and halt. Otherwise, bindings in M_1 forms initial set of branching variables, and branching variables are stored in the branching table. See Figure 2 (a). At first, the branching point is located at the last entry of the branching table.

2. (A branching variable is said to be *alternated* if MSAT had already explored its alternative guess.) If the variable at the branching point was already alternated, move up until non-alternated variable is found. If the root is backtracked from, return unsatisfiable. Otherwise, make alternative guess for the variable at the branching point, and mark the variable as alternated. Let $M_1^{'}$ be the binding of the variables from the top of the branching table to the branching point of the branching table. See Figure 2 (b).
3. Let $F_1^{'}$ be the combined formula of F_1 and $M_1^{'}$ that is simplified. If $F_1^{'}$ is unsatisfiable, go to the step 2. Otherwise, inspect the bindings of the cut variables. If there exist new bindings not present in $M_1^{'}$, let $M_1^{''}$ be $M_1^{'}$ with the new bindings. See Figure 2 (c). Furthermore, generate $F_2^{''}$ by simplifying the combination of F_2 and $M_1^{''}$. If $F_2^{''}$ is satisfiable, return satisfiable. Otherwise, go to the step 2.

When F_1 in the step 1 (or $F_1^{'}$ in the step 3) is satisfiable, the number of entries-depth of the branching path-in the branching table grows. If we can find the partial model with least cut set variables, the searching time can be reduced. Currently, MSAT just inspects the model for F_1 (or $F_1^{'}$) found by the embedded SAT tester.

If an existing tester can output a partial assignment when the input formula is satisfiable, it is well suited for using our second heuristic. MSAT can employ the tester as a subroutine for determining satisfiability in the step 1 & 3.

5.3 A Modified Tester

MSAT was implemented by modifying `2cl` of Tsuji and Van Gelder [VGT95]. `2cl` can detect the equivalent literals. If two unassigned cut variables are equivalent, the equivalence must be added to the testing subformula in addition to the unit clause constraints. Otherwise, MSAT can find M_{F_2} that can violate the equivalence. In this case combining M_{F_1} and M_{F_2} does not result a model for F. Therefore, the program adds the equivalence relation (two binary clauses) between unassigned cut variables into M_1 or $M_1^{''}$. When the program backtracks, a equivalence relation is negated by complementing a literal involved in the equivalence relation.

To reduce the number of assigned cut variables, we introduce the priority concept. The priority of V_1 variables have higher priority value than the priority of V_c variables. The modified `2cl` is restricted to select a branching variables among V_1 variables first. Later, `2cl` branches among the V_c variables when there is no V_1 variables left.

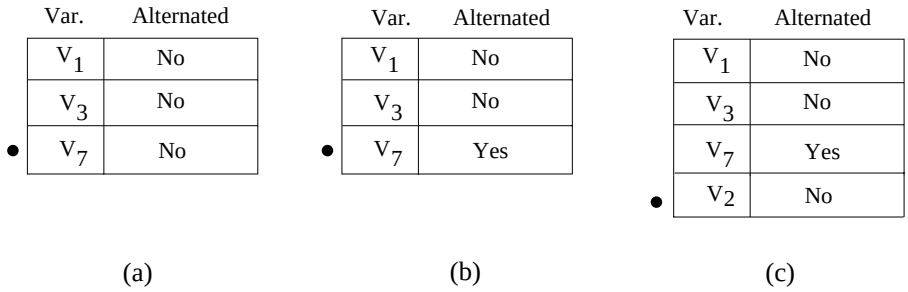

Fig. 2. Illustration of the branching action of MSAT. The first column of the branching table shows the name of binded cut variables, and the second column indicates whether the branching variable was alternated or not. The • symbol indicates a *branching point*. If there exists no satisfying truth assignment with current binding of cut variables, MSAT makes alternative guess for the variable where the branching point is located. Assume that M_1 is $\{v_1, v_3, v_7\}$ at the end of 1 of Section 5.2. (a) M_1 is stored in the branching table. (b) The truth assignment of v_7 is reversed since v_7 was not alternated before, and the resulting $M_1^{'}$ is $\{v_1, v_3, -v_7\}$. (c) Assume that $F_1^{'}$ is satisfiable with a new binding to cut variable $v_2 = false$. In that case, the resulting $M_1^{''}$ is $\{v_1, v_3, -v_7, -v_2\}$. Also, the branching point is updated to the last entry of the table.

6 Experimental Results

This section reports experimental results on difficult formulas from a variety of sources. Nemesis, a test-pattern generation program described by Larrabee[Lar92], produced the first five formulas in Figure 3, from the ISCAS-85 benchmark circuits. The first two formulas are *circuit diagnosis* formulas, and the other three formulas are *single stuck-at* formulas. Most of formulas generated by Nemesis were relatively easy for `2cl`, and did not require partitioning. The last two formulas in Figure 3 are DIMACS benchmark formulas from the second DIMACS Implementation Challenge. All of the seven formulas are unsatisfiable formulas. Before partitioning the input formulas, all the test formulas were simplified using the simplification rules, and the resulting formulas are shown in Figure 3.

Partitioning result of the test formulas is shown in Figure 3, and it was obtained by the implementing linear time algorithm of Fiduccia and Mattheyses [FM82]. If one of the resulting subformula is unsatisfiable by itself, we consider the number of branches taken to be 0. The number of branches taken is shown in the column 9 of the table Figure 3. Both the formula c5315-1 and the formula c5315-3 were determined unsatisfiable at initial step of MSAT. The the size of cut set for the formula ssa2670-127 is 10. Thus there are 1024 possible combination of truth assignments for the cut variables. However, the actual number of combinations inspected by MSAT is 90. Although the number of branches taken by the formula c2670-18 is less than the formula c2670-16, the total CPU time is about 5 times more. It shows that the subformulas of the formula c2670-18 is more difficult than the subformulas of the formula c2670-16 for the given

formula	n	m	n_{V_c}	n_{V_1}	n_{V_2}	m_{F_1}	m_{F_2}	branch taken	CPU time
c5315-1	728	2199	6	316	406	938	1261	0	265
c5315-3	728	2200	6	316	406	938	1262	0	264
c2670-13	606	1642	11	218	377	613	1029	135	2397
c2670-16	626	1642	11	400	195	1089	553	188	1608
c2670-18	626	1642	10	420	176	1203	439	44	8105
pret150-75	150	400	5	83	62	228	172	9	14059
ssa2670-127	449	1246	10	301	138	841	405	90	650

Fig. 3. Test formulas and the resulting partitions. The second column indicates the number of variables, and the third column indicates the number of clauses. The fourth through the eighth column present the result of partitioning. For example, the size of resulting cut set is shown in fourth column. The last column shows the running time for each formula by MSAT. Times are CPU seconds on a Sun SPARCsystem 10/41.

partition.

A comparison of the running time between `2cl` and MSAT is shown in Figure 4. 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 only 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.

The formula pret150-75 took about 4 CPU hours for MSAT, but it only made 9 branch alternation. From this data, we observe that the partitioned subformulas are hard. However, we can also predict that MSAT can do better if we decompose the subformula of pret150-75 further. Currently, the input formula is only partitioned into two subformulas, but we could partition the subformula if the subformula is hard to solve. Therefore, we have some *control* over hard formulas when they are encountered.

7 Conclusion

We have introduced two heuristics that are based on partitioning an input formula. These two heuristic are control programs that use 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 none) modification of the existing SAT tester. This supports our intuition that dealing with subformula can be within the reach of existing SAT testers although the original formula may not be.

formula	2cl guesses	MSAT guesses	2cl runtime	MSAT runtime		
				solver	partition	total
c5315-1	> 78527741	5955	> 86268	233	32	265
c5315-3	533548868	5955	741187	233	31	264
c2670-13	2207532	86341	20950	2214	183	2397
c2670-16	1857622	49018	9547	1608	183	1791
c2670-18	2087018	1132101	19105	8105	188	8293
pret150-75	> 27804607	11461369	> 36000	14057	2	14059
ssa2670-127	477168	11965	1369	554	96	650

Fig. 4. 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. The column under the heading "total" shows the addition of MSAT runtime and partitioning time. Times are CPU seconds on a Sun SPARCsystem 10/41.

Acknowledgments

Both of the authors were in part supported by NSF Grant CCR-9503830. We thank the anonymous referee for many suggestions on improving the paper. We also 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

[CSZ94] P.K. Chan, M.D.F. Schlag, and J.Y. Zien. "Spectral K-way ratio-cut partitioning". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and systems*, 9:1088–1096, Sept. 1994.

[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.

[Don88] W.E. Donath. "Logic Partitioning". In *in Physical Design Automation of VLSI systems, Benjamin/Cummings*, pages 65–86, 1988.

[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.

[GJ79] M. Garey and D.S. Johnson. *"Computers and Intractability: A Guide to Theory of NP-Completeness"*. M.H.Freeman, 1979.

[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.

[Kah91] Andrew B. Kahng. "New Spectral Method for Ratio Cut Partitioning and Clustering". *IEEE Trans. on CAD*, 1991.

[KGV83] S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi. "Optimization by Simulated Annealing". *Science*, 13(220):671–680, May,1983.

[KL70] B.W. Kernighan and S. Lin. "An Efficient Heuristic Procedure for Partitioning Electrical Circuits". In *Bell System Technical J.*, Feb. 1970.

[Lar92] T. Larrabee. "Test Pattern Generation Using Boolean Satisfiability". *IEEE Transactions on Computer-Aided Design*, 11:4–15, January 1992.

[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.

[WC89] Y.C. Wei and C.K. Cheng. "Towards Efficient Hierarchical Designs by Ratio Cut Partitioning". In *IEEE Intl. Conf. on Computer-Aided Design*, pages 298–301, 1989.