Partition-Based Decision Heuristics for Image Computation using SAT and BDDs

Aarti Gupta, Zijiang Yang, Pranav Ashar
NEC USA, CCRL
4 Independence Way, Princeton, NJ 08540
{agupta,jyang,ashar}@nec-lab.com

Lintao Zhang, Sharad Malik
Dept of Electrical Engineering
Princeton University, Princeton, NJ 08540
{lintaoz,sharad}@ee.princeton.edu

Abstract

Methods based on Boolean satisfiability (SAT) typically use a Conjunctive Normal Form (CNF) representation of the Boolean formula, and exploit the structure of the given problem through use of various decision heuristics and implication methods. In this paper, we propose a new decision heuristic based on separator-set induced partitioning of the underlying CNF graph. It targets those variables whose choice generates clause partitions with disjoint variable supports. This can potentially improve performance of SAT applications by decomposing the problem dynamically within the search. In the context of a recently proposed image computation method combining SAT and BDDs, this results in simpler BDD subproblems. We provide algorithms for CNF partitioning – one based on a clause-variable dependency matrix, and another based on standard hypergraph partitioning techniques, and also for the use of partitioning information in decision heuristics for SAT. We demonstrate the effectiveness of our proposed partition-based heuristic with practical results for reachability analysis of benchmark sequential circuits.

1 Introduction

The Boolean satisfiability problem (SAT) has recently received considerable attention in many verification applications, such as equivalence checking [6, 12, 23], as well as model checking [1, 3, 26]. Recently, combining SAT techniques with BDDs has been shown to be effective for image computation with application in state reachability analysis of sequential circuits [13].

A typical implementation for solving SAT uses a branch-and-bound search over the values of all variables, with considerable sophistication in the software engineering of techniques for decision making, implication gathering, and backtracking [19, 22, 28]. Since the SAT problem itself is NP-complete, the effectiveness of any algorithm for solving SAT depends upon the amount of pruning of the search space that it enables. Decision heuristics, i.e. the choice of the SAT variable to branch on, and its value, directly affect the amount of pruning. Many SAT implementations use a Conjunctive Normal Form (CNF) representation of the Boolean formula. This has led to the development of many decision heuristics based on the frequency of appearance of variables in unsatisfied (or all) clauses, sometimes giving preference to smaller clauses in order to facilitate implications [17]. In this paper, we focus on decision heuristics targeted at decomposing the overall problem into smaller, unrelated, partitions.

1.1 Motivation

A recently reported method for image computation uses SAT search as a disjunctive decomposition of the overall search for image solutions into multiple subproblems, each of which is handled by using a standard BDD-based image computation algorithm [13]. In this context, SAT decision heuristics affect not only the pruning of the search space in SAT, but also the complexity of dynamically generated BDD subproblems.

We propose a new decision heuristic for SAT, based on separator-set induced partitioning of the underlying CNF graph. We use separators instead of minimum cutsets for partitioning, because there exist small separators for graphs that do not have bipartitions with small cutsize [16]. The decision heuristic is targeted at those variables whose choice dynamically results in clause partitions with disjoint variable supports. Since disjoint subproblems contribute to search complexity additively, rather than multiplicatively, this heuristic can potentially improve performance in many applications of SAT, especially those where a large part of search space needs to be explored. Specifically, for the image computation problem, use of the proposed decision heuristic in SAT, leads to simpler BDD subproblems. This is because BDD image computations with disjoint conjunctive partitions are less likely to blow up in size, in comparison to those with connected partitions.

In this paper, we describe two different methods for partitioning the CNF graph. One is based on the MLP (Minimal Lifetime Permutation) approach proposed by Moon *et al.* [20], and the other is based on use of a standard hypergraph partitioning package called hMETIS [14]. We also provide a simple algorithm which uses the partition information to assign weights to all CNF variables, which can be combined with standard SAT decision heuristics.

The benefit of complementing a purely functional approach based on BDDs with structural information captured by SAT is crucial in improving performance of image computation. We present practical results on benchmark circuits demonstrating this impact. We show that the use of our proposed heuristic consistently improves the performance for reachability analysis, in some cases enabling the prototype tool to reach more states than possible without the use of this heuristic.

1.2 Related Work

There has been some effort in exploring the benefits of partitioning for generic SAT applications [18], but this was restricted to the *detection* of partitions as they arise dynamically within the search, and no effort was made to actually derive such partitions. There has been some independent work in the SAT community on use of partitioning methods similar to ours, in order to improve the efficiency of the SAT solver [2]. However, this effort is not directly targeted at deriving good decision heuristics, and they do not provide any empirical results on practical problems. Along another related line, it has been conjectured that the degree of difficulty of a given SAT problem is related to the information "bandwidth" of the problem [24, 15], i.e. the greater the connectivity between variables, the more difficult the problem is likely to be. Again, this observation can be used to justify choosing decision variables which partition the problem into low bandwidth (or disjoint) partitions.

In terms of image computation itself, there have been many efforts aimed at exploiting circuit structure information for a pure BDD-based image computation [10]. For example, heuristics for clustering and ordering are based on analysis of shared variable support sets between next-state bit relations and the input state set [25]. In particular, Moon *et al.* proposed the MLP algorithm for a dependency matrix representation to obtain a Bordered Block Triangular form, which is particularly suited for deriving a good conjunction schedule [20]. We use this form directly for partitioning, described later in the paper. They also identify existing connected components for decomposition, but again, there is no effort to actively derive such decompositions dynamically. A recent technique [9] also uses graph partitioning methods to find a good order for clusters. Note that many of these heuristics capture the benefits of partitioning on the underlying circuit structure, which is similar to our goal. However, none of these methods actually use SAT. Therefore, the specific partitioning methods we use for CNF graphs, as well as their use for choosing decision heuristics within SAT, are novel in our approach.

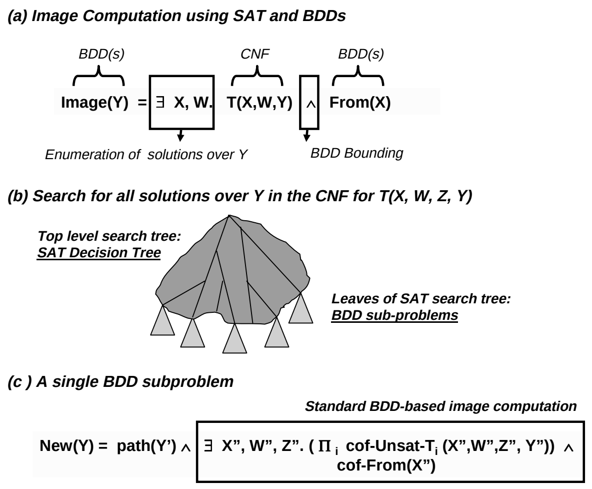
(a) Image Computation using SAT and BDDs
BDD(s)
CNF
BDD(s)
Image(Y) = ∃ X, W. T(X,W,Y) ∧ From(X)
Enumeration of solutions over Y
BDD Bounding
(b) Search for all solutions over Y in the CNF for T(X, W, Z, Y)
Top level search tree:
SAT Decision Tree
Leaves of SAT search tree:
BDD sub-problems
(c) A single BDD subproblem
Standard BDD-based image computation
New(Y) = path(Y') ∧ ∃ X", W", Z". (Π i cof-Unsat-Ti (X",W",Z", Y")) ∧ cof-From(X")

Figure 1: Image Computation using SAT and BDDs

The rest of the paper is organized as follows. In Section 2, we provide the necessary background for image computation based on SAT and BDDs. Our algorithms for partitioning CNF graphs, and our SAT decision heuristic based on this information is described in Section 3. Experimental results demonstrating the benefits of our heuristic are presented and discussed in Section 4, followed by conclusions.

2 SAT-based Image Computation

Historically, symbolic state space traversal [7, 11] has relied on efficient algorithms based on BDDs [5] for carrying out an image computation, shown below:

$$Image(Y) = \exists_{X,W}\ T(X,W,Y) \wedge From(X) \qquad (1)$$

Here, X/Y denote present/next state variables, W denotes primary input variables, T denotes the transition relation, and $From$ denotes the input state set. BDDs are used to represent the characteristic function of the transition relation, as well as the input/image sets. As an example application, the set of reachable states can be computed by starting with a set $From$ which denotes the set of initial states of a system, and using image computation iteratively, until a fixpoint is reached. The BDD-based approaches work well when it is possible to represent the sets of states and the transition relation (as a whole, or in a usefully partitioned form) using BDDs. Unfortunately, BDD size is very sensitive to the number of variables, variable ordering, and the nature of the logic expressions being represented.

Recently, an integration of SAT and BDDs has been proposed for image computation [13]. A pictorial representation of various features of this method is shown in Figure 1. As shown in Part (a), state sets are represented by BDDs, and the transition relation is represented as a CNF formula. All image solutions over Y are enumerated using a backtracking search algorithm for SAT which operates over the CNF formula for T. Within this search, the BDD for $From(X)$ is used as a constraint (called BDD Bounding), where any partial assignment over the X variables that does not satisfy $From(X)$ leads to immediate backtracking within SAT. As shown in Part (b), rather than using SAT to enumerate each solution all the way down to a leaf, BDD-based subproblems are invoked at intermediate points within the SAT search. This allows a symbolic, rather than explicit, enumeration of all solutions in the subtree rooted below that point. In a sense, this approach can be regarded as SAT providing a disjunctive decomposition of the image computation into many BDD subproblems. Each of the BDD subproblems involves a standard image computation as shown in Part (c), where the BDDs for the conjunctive partition are generated on-the-fly from unsatisfied clauses of the CNF for T.

3 CNF Partitioning

In this section, we describe our algorithms for two partitioning methods and the decision heuristic for SAT. The basic idea is to use partitioning methods on a CNF formula to obtain a good *separator*, i.e. a set of clauses which separates the remaining clauses into two sets with no common variables. In other words, we obtain three partitions of the entire set of clauses – called left, right, and separator, such that the left and right partitions do not share any variables. We focused on the use of separators for partitioning, rather than using a minimum cutset, because there exist graphs which have a small separator, but do not have bipartitions with small cutsets [16]. For example, a star graph with a central node connected to *n* other nodes, has a one-node separator, but an *O(n)* cutset. Furthermore, we use a recursive partitioning scheme, whereby any partition is considered for further re-partitioning if its size is above a certain threshold.

Note that a three-way partition of the set of clauses does not necessarily correspond to a partition of the support variables. This is shown pictorially for the general case in Figure 2, which shows the shared and the private (non-shared) variables for each of the three clause partitions.

During SAT, we give preference to making decisions on the separator variables, in order that the remaining search can be performed over the disjoint left and right partitions. Note that it is sufficient, but not necessary, to choose only shared variables in order to obtain partitions with disjoint variable supports. For example, it may be possible to assign values to private separator variables in order to satisfy all clauses in the separator partition. This still leads to the remaining problem being disjoint in terms of the left and right clause partitions. Our approach based on CNF partitioning allows us to take into account not only the variable

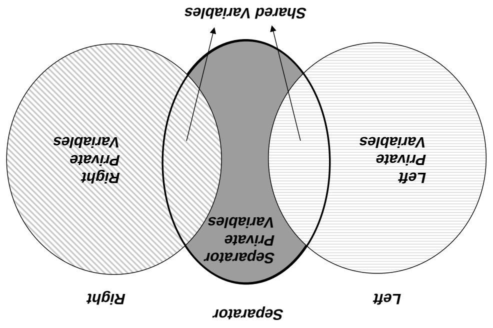

Figure 2: Variables of Clause Partitions

support sets, but also the constraints on their values, i.e. the clauses. This is in contrast to other approaches which may consider only variable support sets for partitioning.

3.1 Dependency Matrix Method

The first partitioning method is based on the work by Moon *et al.* [20]. They use a dependency matrix representation of conjunctive partitions (rows) and their variable support sets (columns). The rows and columns of this matrix are then permuted using the MLP algorithm to obtain a Bordered Block Triangular form. The basic algorithm consists of working iteratively on the active region of the matrix, by choosing a column which intersects the maximum number of shortest rows, moving it to the leftmost position, and shrinking the active region to exclude it. This form is used to obtain a good conjunction schedule for purely BDD-based image computation.

We use a similar matrix representation for capturing the dependencies between clauses (rows) and variables (columns) of the CNF formula to be partitioned. In our implementation, we do not use all CNF clauses representing the transition relation, since this sometimes resulted in clauses from the same gate being in different partitions. Instead, we consider only the minimum number of clauses for each gate of the circuit such that all variable dependencies are captured. We also add a row to denote the dependency of the input set BDD on various variables. Next, we use the basic MLP algorithm proposed by Moon *et al.*, with minor modifications, to obtain the Bordered Block Triangular form. The modifications consist of several rules to break ties whenever there are multiple column candidates that can be moved to the left. In particular, preference is given to those columns with a higher "affinity" with the inactive region, i.e. with more number of entries in the inactive matrix.

Next, we use this matrix to choose a good *separator*. This is done by choosing a good separator variable, as shown in the pseudo-algorithm in Figure 3. Basically, each variable (column) is considered as a candidate for separation, because it divides all clauses into three sets – those with all variables to its left, those with all variables to its

```
choose_separator_variable()
{
   max = 0;
   for (each column i) {
      set varsInL, varsInSep, varsInR;
      for (each row j) {
         if (all entries in j are to the left of i)
            varsInL = varsInL UNION (vars in row j);
         else if (all entries in j are to the right of i)
            varsInR = varsInR UNION (vars in row j);
         else
            varsInSep = varsInSep UNION (vars in row j);
      } // end of loop on j
      set PrivateVarsInL = varsInL - varsInSep;
      set PrivateVarsInR = varsInR - varsInSep;
      merit = (|PrivateVarsInL| * |PrivateVarsInR|) /
            |varsInSep|;
      if (merit > max) {
         max = merit;
         sep_var = i;
      }
   } // end of loop on i
   return sep_var;
}
```

Figure 3: Choosing a Separator Using Dependency Matrix

right, and those with variables both to its left and right. The last set of clauses is actually the separator set associated with this variable. The figure of merit we use for each variable is targeted at maximizing the number of private variables in each of the left and right partitions, while minimizing the total number of variables in the separator itself. This matrix-based separator algorithm is used at every level of the recursive partitioning scheme to yield three disjoint partitions of clauses at each level.

3.2 Graph Partitioning Method

We have explored another CNF partitioning method based on standard hypergraph partitioning techniques. The CNF graph we consider consists of nodes denoting clauses, and hyperedges denoting variables of the CNF formula. The CNF formula we use is the same as for the Dependency Matrix Method, and we also add a node to the CNF graph denoting the input set BDD. We consider weighted graphs, where each hyperedge has a weight equal to the number of nodes it connects, i.e. the number of clauses that the corresponding variable appears in.

Though polynomial time algorithms exist for finding minimum separators based on maxflow-mincut network algorithms, the problem of finding separators which yield balanced partitions is NP-hard [16]. For partitioned CNF graphs, since the overall complexity of the SAT solver is affected by the sizes of the individual partitions as well, our main interest is in solving the latter problem. For this, we used a publicly available package called hMETIS [14], which is known to perform well on practical problems. However, the hMETIS package cannot be used directly to find good separators. Instead, we use it to find a minimum-weight cutset of the given graph, where the cutset partitions the graph into unconnected components. The minimum-weight cutset consists of hyperedges, i.e. variables, and the separator partition is defined as all clauses that these variables appear in. Note that while this is not equivalent to finding a good separator, minimizing the weights on the hyperedges does tend to give small separators. The variable support set of the separator clauses defines the set of separator variables, which can be larger than the set of cutset variables in general. By definition, the two node partitions defined by the cutset do not share any hyperedges, i.e. they correspond to the left and right partitions of clauses which do not share any common variables.

Again, the graph partitioning algorithm is used at each level of the recursive partitioning scheme, such that each terminal partition is less than a certain threshold. For this reason, we also allow unbalanced partitions, with up to a 25% unbalance factor. (Using a higher unbalance factor results in many trivial terminal partitions.)

3.3 Using Partitions for Decision Heuristics

The result of the partitioning method is a partition tree, such that the size of each terminal partition is less than a certain threshold. This partition tree is used to assign a weight to each variable in the CNF representation of the SAT problem. This partition-based weight is used as a multiplicative factor for the weight/rank computed for each variable using standard SAT heuristics, e.g. DLIS [19].

Our weight assignment algorithm works as follows: we would like to favor decisions on separator variables, in order that the remaining search can be performed over the disjoint left and right partitions. Therefore, we assign separator terminals a weight of 2, and other terminals a weight of 1. Recall from Figure 2 that some variables of the separator partition might be shared, while others are private. Again, we give more preference to shared separator variables, rather than private separator variables. This is because shared separator variable assignments are more likely to lead earlier to disjoint partitions. Therefore, for each variable, its weight is obtained by adding the contribution of each terminal partition that the variable appears in. This ensures that shared variables get more weight than non-shared variables.

For example, consider the partition tree shown in Figure 4, where all terminal partitions are labeled by the support variables, and the boxed numbers denote the weights assigned to the terminal partitions – weight 2 for separators, weight 1 otherwise. The resulting weight assignment

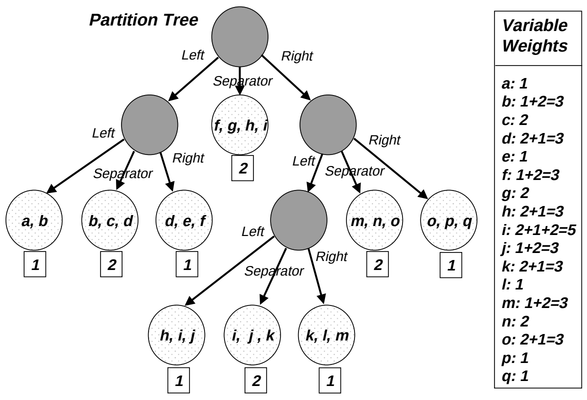

Figure 4: Example Partition Tree

for each variable is shown in the box to the right of the partition tree.

We also experimented with alternative weight assignment strategies. In one, the weight assigned to each terminal partition decreased according to increased depth from the top of the tree. In another, the weight also depended on the size of the partition, with smaller partitions being given higher weight, in order to achieve disjoint partitions as soon as possible in the SAT search. However, none of these variations worked as well as, or improved upon, the simple scheme described above.

4 Experimental Results

There has been significant progress made in symbolic reachability analysis in recent years. We compare the SAT-based image computation with and without the partition-based decision heuristic, to state of the art techniques in VIS [4], a public domain tool. Our prototype implementation of the SAT-based image computation algorithm uses the GRASP SAT solver [19] and the CUDD BDD package [27], and has been integrated within VIS. All reported experiments were run on an UltraSparc machine, with a 296 MHz processor, 1 GB memory. For most experiments a time limit of 100K seconds was used, and the times indicated in the table are for the last reachability step completed within this time limit. For VIS[1], we used the default options in all experiments (partition threshold=5000, frontier method for building partition MDDs, iwls95 image method, and image cluster size=5000) Dynamic variable reordering was enabled throughout all experiments, and good orders on the state and primary input variables (when available) were used initially. (Note that the SAT-based image computation requires additional ordering on the internal variables also, which appear in the CNF formula.)

[1] These experiments were conducted with VIS version 1.3, since version 1.4 was not available at that time.

4.1 Partitioning

Results of the two partitioning methods for the ISCAS benchmark circuits are shown in Table 1. The name of the circuit appears in Column 1, and number of latches in Column 2 (marked #L). The number of CNF variables and clauses are shown in Column 3 (marked #V / #C, respectively). Columns 4 through 7 show the results for the Dependency Matrix method – the depth of the partition tree, the number of terminal partitions, size of the biggest terminal partition (#v / #c), and the CPU time required for partitioning respectively. Columns 8 through 11 show corresponding results for the Graph Partitioning method. For these experiments, the input set BDD corresponds to the initial state set. For reachability analysis, we perform CNF partitioning dynamically at each iteration, taking into account the dynamically changing input set BDD.

As can be seen from Table 1, both partitioning methods are quite efficient in obtaining multi-depth partition trees for all benchmark circuits. In general, for the same threshold size constraint (different across circuits), the Graph Partitioning method (based on hMETIS) results in smaller depth trees, and less number of terminal partitions than the Dependency Matrix method. However, as results in the next section show, this did not always lead to improved results for reachability analysis.

4.2 Reachability Analysis

Results for reachability analysis on the benchmark circuits are shown in Table 2. The name of the circuit appears in Column 1, and and a “(C)” after the name indicates that at least one method was able to complete the traversal, i.e. a fixpoint was reached. Columns 2, 3, and 4 show the number of steps completed (n), the CPU time (in seconds), and the Peak number of BDD nodes (in Millions) for standard VIS [4], which uses only BDDs for image computation. The remaining columns report the results for the SAT-based image computation which uses both SAT and BDDs. Columns 5, 6, and 7 report these numbers without the use of partition-based decision heuristic. Columns 8, 9, and 10 report these numbers with the use of partition-based decision heuristic using the Dependency Matrix (DM) method, while Columns 11, 12, and 13 report these numbers for the Graph Partitioning (GP) method using hMETIS. The CPU times reported include the time spent on obtaining the partition tree dynamically for each reachability step. Finally, Column 14 reports the improvement obtained by using the best partitioning method (DM or GP) in comparison to not using the partition-based decision heuristic at all. The improvement is reported either as an increase in the number of reachability steps, or as a speedup factor when the number of reachability steps completed are the same, i.e. (time without partition heuristic)/(time with partition heuristic).

Note from the last column of Table 2 that for all cir-

Name	#L	CNF Size	Dependency Matrix Method				Graph Partitioning Method			
		#V / #C	Tree Depth	# Tree Terminals	Biggest Size #v / #c	Time (s)	Tree Depth	# Tree Terminals	Biggest Size #v / #c	Time (s)
s1269	37	488 / 1308	6	11	181 / 342	2.7	3	7	174 / 282	1.5
s1423	74	589 / 1494	6	13	189 / 345	4.1	4	9	155 / 205	1.7
s3271	116	1273 / 3387	6	21	182 / 354	12.7	6	21	187 / 438	5.3
s3330	132	885 / 2214	7	15	193 / 325	8.7	5	13	200 / 363	2.9
s3384	183	1187 / 2853	7	21	217 / 456	9.7	6	17	239 / 154	3.1
s4863	88	1887 / 5250	7	21	286 / 642	46.7	5	19	256 / 678	7.5
prolog	136	921 / 2322	7	13	193 / 242	9.5	4	11	192 / 259	2.8
s5378	164	1234 / 3085	8	23	289 / 611	17.8	5	11	288 / 328	3.3
s6669	231	2440 / 6302	8	27	280 / 669	55.0	7	23	297 / 188	9.4
s9234.1	211	2316 / 6548	4	15	484 / 1012	38.6	5	15	445 / 725	9.8
s13207.1	638	3464 / 8773	9	43	520 / 974	110.9	10	35	526 / 1159	16.2

Table 1: Partitioning Results for Benchmark Circuits

cuits, except s5378, the use of at least one partitioning method improves the performance of the SAT-based image computation *in comparison to use of no partitioning method*. The improvement in performance for the same number of steps is up to factor of 6, or more number of reachability steps are completed. For one circuit, s4863, the use of the GP method allowed a complete traversal, which could not be done earlier. In our experiments, the CPU times were dominated by the time needed to solve the BDD sub-problems. With the use of partition-based decision heuristics in SAT, these BDD sub-problems are considerably simplified because they consist of image computation over more loosely-connected (sometimes disjoint) BDD relations. Therefore, these improvements clearly indicate the benefit of adding partition-based information in SAT for this application. Furthermore, both partitioning methods perform fairly well, each contributing to best improvement in about half of the circuits. At this time, our experiments are not conclusive in terms of characterizing which method is better suited for which kind of circuits (or partition trees).

In comparison to standard BDD-based image computation, note again from the table that SAT-based image computation is able to outperform standard VIS for many of the benchmark circuits, sometimes by an order of magnitude. For some of the larger circuits, such as s9234.1 and s13207.1, the SAT-based methods are not as good as VIS due to the large number of variables in the CNF representation, in contrast to a BDD-based transition relation representation. However, there is scope for improvement in this direction by use of clustering to pre-quantify variables statically, and we are currently exploring such methods.

Other researchers have also recently reported improvements for these benchmark circuits in comparison to standard VIS by use of better conjunction/ quantification schedules [9, 20, 21]. However, since these enhancements are not publicly available, we could not conduct experiments within our environment, and it is difficult to make a fair comparison. (Note that our implementation of the dependency matrix method of Moon *et al.* works on the CNF graph, not on the next-state bit relations as in their original work.) We are also aware of other prioritized (non breadth-first) traversal techniques which have shown good results for reachability analysis. (See a recent paper [8] for more details and other references). However, the target application for our current paper is image computation, and we have focused on pure breadth-first traversal as a good indicator of its performance. It is our belief that just like pure BDD-based image computation has seen many advances with sophisticated heuristics and better engineering, there is further potential to improve SAT-based image computation, which complements the benefits of BDDs by incorporating domain-specific knowledge through SAT.

4.3 Separator-set Induced Partitioning

Recall from Section 3.2 that in our CNF graph partitioning method based on hMETIS, each hyperedge (denoting a CNF variable) has a weight, which represent the number of clauses the variable appears in. We then used hMETIS to find a minimum-weight cutset, and associated the separator partition with all clauses that the cutset variables appeared in. Our motivation for using edge weights was to minimize the number of clauses in the separator, and thereby potentially minimize the total number (shared and private) of separator variables. (There is no direct way to use hMETIS to find good separators.)

We also explored the use of hMETIS without edge weights, i.e. all hyperedges are given a weight of 1. In this case, hMETIS again obtains minimum cutsets, but these do not necessarily correspond to good separators. This is because only the number of shared separator variables is minimized, with no effort to minimize either the number

Circuit	VIS (Standard)			SAT+BDD Image									
				No Partition			DM Method			GP Method			Improvement
	n	Time (s)	Peak (M)	n	Time (s)	Peak (M)	n	Time (s)	Peak (M)	n	Time (s)	Peak (M)	
s1269 (C)	10	3269	6.7	10	2164	0.8	10	2000	1.5	10	1685	1.7	1.28 (GP)
s1423	11	8791	6.5	13	13151	12	15	24933	30.6	14	33013	17.2	2 more (DM)
s3271 (C)	17	17933	6	17	14036	1.4	17	9700	2.2	17	12791	2.4	1.45 (DM)
s3330 (C)	9	20029	6	9	2029	1	9	4351	1.3	9	748	0.8	2.71 (GP)
s3384	4	24844	3.6	6	7801	0.5	6	3843	0.6	7	15307	2.9	1 more (GP)
s4863 (C)	3	3592	6	1	1014	0.6	5	39250	1.6	4	9488	0.7	4 more (DM)
prolog (C)	4	22099	5	9	4697	2.8	9	858	0.7	9	726	1.1	6.47 (GP)
s5378 (C)	8	57986	22.7	45	60547	2.6	45	95960	2.5	45	82644	1.6	0.73 (GP)
s6669	3	505	1	2	549	0.3	2	317	0.3	2	542	1.3	1.73 (DM)
s9234.1	9	11577	7.6	9	22777	7.7	11	96455	17.3	9	15769	1.4	2 more (DM)
s13207.1	14	28600	7.5	10	8340	0.8	8	4910	0.8	11	13548	2.3	1 more (GP)

Table 2: Reachability Results for Benchmark Circuits

of separator clauses, or the total number of separator variables. Therefore, for the same size threshold for recursive partitioning, we obtained different partition trees with no edge weights. These were then used in the same way as described earlier for computing decision heuristics in SAT.

The results for reachability analysis on some circuits using hMETIS with and without edge weights are shown in Table 3. In this table, Columns 2, 3, and 4 report the number of steps completed (n), the CPU time (in seconds), and the Peak number of BDD nodes (in Millions) for the hMETIS-based partitioning heuristic with edge weights, while Columns 5, 6, and 7 report these numbers without edge weights. Note that in all circuits, the performance of reachability analysis suffers when hMETIS is used without edge weights. This highlights the benefit of using good separators for partitioning, instead of using minimum cutsets alone.

5 Conclusions

We have proposed a decision heuristic for SAT which favors those variables whose choice results dynamically in disjoint variable supports for clause partitions in the underlying CNF graph. When used in combination with BDDs for image computation, this heuristic has the effect of simplifying the associated BDD subproblems, because the conjunctive partitions are more loosely coupled than before (and are disjoint in some cases). We have provided algorithms for two CNF partitioning methods – one based on use of a clause-variable dependency matrix, and another based on use of a standard package for hypergraph partitioning. We have also described details of using this partitioning information to modify standard decision heuristics in SAT. We have presented practical results for reachability analysis on a number of benchmark circuits, which show a consistent performance improvement due to our partition-based decision heuristic, and also demonstrate the benefits of a separator-set induced partitioning method. We are currently exploring the use of this heuristic for general SAT applications.

References

[1] P. A. Abdulla, P. Bjesse, and N. Een. Symbolic reachability analysis based on SAT-solvers. In *Tools and Algorithms for the Analysis and Construction of Systems (TACAS)*, 2000.

[2] E. Amir and S. McIlraith. Partition-based logical reasoning. In *Proc. 7th International Conference on Principles of Knowledge Representation and Reasoning*, 2000.

[3] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Analysis and Construction of Systems (TACAS)*, volume 1579 of *LNCS*, 1999.

[4] R. K. Brayton et al. VIS: A system for verification and synthesis. In R. Alur and T. Henzinger, editors, *Proc. Int. Conf. on Comput.-Aided Verification*, volume 1102 of *LNCS*, pages 428–432, June 1996.

[5] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Tran. on Comp.*, C-35(8):677–691, Aug. 1986.

[6] J. Burch and V. Singhal. Tight integration of combinational verification methods. In *Proc. Int. Conf. on Comput.-Aided Design*, pages 570–576, 1998.

[7] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Tran. on CAD of Integrated Circ. and Sys.*, 13(4):401–424, Apr. 1994.

Circuit	With Edge Weights			Without Edge Weights		
	n	Time (s)	Peak (M)	n	Time (s)	Peak (M)
s1269	10	1685	1.7	10	1985	1.7
s3271	17	12791	2.4	13	25767	4.2
s3330	9	748	0.8	9	3261	2.5
prolog	9	726	1.1	9	4925	1.4
s4863	4	9488	0.7	4	32843	0.8
s5378	45	82644	1.7	13	88690	2.0

Table 3: Reachability Results for Partitioning with hMETIS

[8] G. Cabodi, P. Camurati, and S. Quer. Biasing symbolic search by means of dynamic activity profiles. In *Proc. Conference on Design Automation and Test Europe (DATE)*, Mar. 2001.

[9] P. Chauhan et al. Efficient image computation. Personal Communication.

[10] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[11] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines using symbolic execution. In *Proc. Int. Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 365–373. Springer-Verlag, June 1989.

[12] A. Gupta and P. Ashar. Integrating a Boolean satisfiability checker and BDDs for combinational verification. In *Proc. VLSI Design Conference*, Jan. 1998.

[13] A. Gupta, Z. Yang, A. Gupta, and P. Ashar. SAT-based image computation with application in reachability analysis. In *Proc. Conference on Formal Methods in Computer-Aided Design*, Nov. 2000.

[14] G. Karypis et al. hMETIS: Serial hypergraph and circuit partitioning. http://www-users.cs.umn.edu/˜karypis/metis/hmetis.

[15] J. Kukula. When is SAT hard? Presented at Dagstuhl Seminar *Design and Test* on *BDDs versus SAT*, Schloss Dagstuhl, Germany, Jan. 2001.

[16] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley & Sons, England, 1990.

[17] J. P. Marques-Silva. *Search Algorithms for Satisfiability Problems in Combinational Switching Circuits*. PhD thesis, EECS Department, University of Michigan, May 1995.

[18] J. P. Marques-Silva and A. L. Oliveira. Improving satisfiability algorithms with dominance and partitioning. In *IEEE/ACM International Workshop on Logic Synthesis*, May 1997.

[19] J. P. Marquez-Silva. Grasp package. http://algos.inesc.pt/˜jpms/software.html.

[20] I.-H. Moon, G. Hachtel, and F. Somenzi. Border-block triangular form and conjunction schedule in image computation. In *Proc. Conference on Formal Methods in Computer-Aided Design*, Nov. 2000.

[21] I.-H. Moon, J. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: The question in image computation. In *Proc. Design Automation Conf.*, pages 23–28, June 2000.

[22] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Engineering a (super?) efficient SAT solver. In *Proc. Design Automation Conf.*, June 2001.

[23] V. Paruthi and A. Kuehlmann. Equivalence checking combining a structural SAT-Solver, BDDs and simulation. In *Proc. Int. Conf. on Comput. Design*, Oct. 2000.

[24] M. R. Prasad, P. Chong, and K. Keutzer. Why is ATPG easy? In *Proc. Design Automation Conf.*, pages 22–28, 1999.

[25] R. K. Ranjan, A. Aziz, R. K. Brayton, B. F. Plessier, and C. Pixley. Efficient BDD algorithms for FSM synthesis and verification. In *International Workshop for Logic Synthesis*, May 1995. Lake Tahoe, CA.

[26] M. Sheeran, S. Singh, and G. Stalmarck. Checking safety properties using induction and a SAT-Solver. In *Proc. Conference on Formal Methods in Computer-Aided Design*, Nov. 2000.

[27] F. Somenzi et al. CUDD: University of Colorado Decision Diagram Package. http://vlsi.colorado.edu/˜fabio/CUDD/.

[28] H. Zhang. SATO: an efficient propositional prover. In *International Conference on Automated Deduction*, number 1249 in LNAI, pages 272–275, 1997.