Using Cutwidth to Improve Symbolic Simulation and Boolean Satisfiability *

Dong Wang Edmund Clarke
Carnegie Mellon University
Pittsburgh, PA 15213
{emc,dongw}@cs.cmu.edu

Yunshan Zhu James Kukula
Synopsys Inc.
MountainView, CA 94043
{yunshan,kukula}@synopsys.com

Abstract

In this paper, we propose cutwidth based heuristics to improve the efficiency of symbolic simulation and SAT algorithms. These algorithms are the underlying engines of many formal verification techniques. We present a new approach for computing variable orderings that reduce CNF/circuit cutwidth. We show that the circuit cutwidth and the peak number of live BDDs during symbolic simulation are equal. Thus using an ordering that reduces the cutwidth in scheduling the gates during symbolic simulation can significantly improve both the runtime and memory requirements. It has been shown that the time complexity of SAT problems can be bounded exponentially by the formula cutwidth and many practical circuits has cutwidth logarithmic of the size of the formulas. We have developed cutwidth based heuristics which in practice can speed up existing SAT algorithms, especially for SAT instances with small cutwidth. We demonstrate the power of our approach on a number of standard benchmarks.

1 Introduction

Symbolic simulation [3] encodes the possible behaviors of a circuit as Binary Decision Diagrams; thus, in one run of symbolic simulation, it is possible to cover many runs of a traditional simulator. Symbolic simulation involves constructing BDDs for the internal signals of a circuit in a bottom-up manner. The BDD for an internal signal cannot be deleted until the BDDs for all signals in the immediate fanout of the signal have been computed. It is often observed that the final BDDs are quite small compared to the intermediate BDDs during the computation. Therefore, it is important to reduce the size of intermediate BDDs as much as possible. The order in which circuit signals are traversed can greatly affect the efficiency of symbolic simulation. Compared to simple breadth-first-search (BFS) or depth-first-search (DFS), an ordering with smaller cutwidth can significantly reduce the number of internal BDDs and improve the overall performance of symbolic simulation.

Recently, a number of efficient SAT algorithms have been proposed [10, 13].These algorithms can handle practical problems with thousands of variables. On the other hand , these algorithms may exhibit poor performance on small examples involving only tens of variables. This phenomenon suggests that the length of an input formula or the number of variables in the formula may not be an accurate measure of the "hardness" of a SAT problem. Exactly the same arguments can be applied to other problems in formal verification such as model checking and reachability analysis. Previous work [2, 8, 12] suggests that cutwidth is often a good measure of difficulty in formal circuit verification.

Prasad, Chong and Keutzer [12] observed that ATPG is an NP-complete problem and yet is often efficiently solvable in practice. Their paper contains a theoretical analysis of ATPG techniques based on SAT, and shows that ATPG complexity is bounded exponentially by the cutwidth of the circuit. The paper also contains a set of experiments which shows that many practical circuits have logarithmic cutwidth with respect to circuit size.

Berman [2] established a bound on the size of the BDD for a circuit. If the BDD variable ordering is consistent with a topological ordering of the circuit, then the BDD size is bounded by $n \times 2^w$, where n is the number of primary inputs to the circuit and w is the cutwidth of the given topological ordering. McMillan [8] generalized Berman's result. He showed that for a given ordering of circuit nodes with *forward cutwidth* w_f and *backward cutwidth* w_r, the BDD size is bounded by $n \times 2^{w_f 2^{w_r}}$.

Since cutwidth is a good measure of circuit complexity, it is natural to use cutwidth in heuristics for guiding searches in formal verification engines. In this paper, we present techniques for efficiently computing the variable orderings that reduce the cutwidth of a circuit or a boolean formula. We also show that cutwidth based heuristics can be used to improve the performance of symbolic simulation and SAT.

In [6], a dynamic programming algorithm is given for

*This research is sponsored by the Gigascale Research Center (GSRC), the National Science Foundation (NSF) under Grant No. CCR-9505472.

propositional satisfiability. This algorithm has polynomial complexity for SAT problems with logarithmic cutwidth. A decision procedure for quantified boolean formula (QBF) is given in [11]. As a special case, the algorithm can be used for propositional satisfiability. The algorithm is efficient for solving "long and thin" circuits, i.e., circuits with small cutwidth. Rather than proposing new SAT algorithms, our approach computes a variable ordering that reduces the cutwidth and uses this ordering to guide variable splitting in existing state of the art SAT engines, like GRASP [13]. In [9], the metric *active life time* is defined and used to order the partitioned transition relations for image computation. Since active life time is closely related to the concept of cutwidth, this suggests that cutwidth based heuristics may be used to order conjunctive partitions of transition relations. Recently, [1] uses an existing layout tool to generate SAT decision orderings and BDD variable orderings.

The rest of this paper is structured as follows. In section 2, we introduce the concept of cutwidth for a CNF formula and for a circuit. In section 3, we present algorithms for computing variable orderings that reduce cutwidth. In section 4, we show how to exploit these variable orderings to speedup symbolic simulation and SAT procedures. We present some experimental results in section 5 and conclude with some directions for future research.

2 Terminology

An undirected *hypergraph* $G(V, E)$ is an extension of an undirected graph, where V is the set of vertices and $E \subseteq \mathcal{P}(V)$. Each *hyperedge* $e \in E$ contains a set of vertices $v \in V$. Given an undirected hypergraph $G(V, E)$, $\vartheta : V \rightarrow \{1, 2, \ldots, \|V\|\}$ will be a bijection that linearly orders the vertices. A vertex v is said to have position i under ϑ if and only if $\vartheta(v) = i$. The range of a hyperedge e under ϑ is $[l_e, h_e)$, where l_e is the minimum vertex position and h_e is the maximal vertex position. The *cut set* $cset_i(G, \vartheta)$ at position i, where $1 \leq i \leq \|V\|$ is the set of hyperedges whose range include i.

$$cset_i(G, \vartheta) = \{e \in E \mid l_e \leq i < h_e\}$$

The cutwidth $c_i(G, \vartheta)$ at position i is the cardinality of the cut set at position i, i.e., $c_i(G, \vartheta) = \|cset_i(G, \vartheta)\|$. The cut set $cset_v(G, \vartheta)$ at node $v \in V$ is the cut set at position $\vartheta(v)$. The cutwidth $c_v(G, \vartheta)$ at node $v \in V$ is defined similarly.

Definition 2.1 Hypergraph Cutwidth
Given an undirected hypergraph $G(V, E)$ and a bijection $\vartheta : V \rightarrow \{1, 2, \ldots, \|V\|\}$, the cutwidth $c(G, \vartheta) = max_{v \in V} c_v(G, \vartheta)$, i.e., the maximal cut width among all vertices.

A propositional formula is represented in clause form. Each clause contains a set of literals. Each literal can be a variable or its negation. The function $unsigned(C)$ of a clause C contains the set of variables of C, but without their signs. For example, if $C = \{\neg a,\ b,\ \neg d\}$, then $unsigned(C) = \{a,\ b,\ d\}$.

Definition 2.2 CNF Formula Cutwidth
Each CNF formula $\psi(V, E)$ will have a corresponding undirected hypergraph $G(V, E')$, where $E' = \{unsigned(c) \mid c \in E\}$. The cutwidth of the formula ψ is defined as the cutwidth of the undirected hypergraph G.

In symbolic simulation, the traversal of gates must satisfy the constraints imposed by the structure of a circuit. In particular, the BDDs of the node in the fanin of a gate must be computed before the BDDs of the nodes in its fanout. To model the structural constraints, we introduce the concept of a directed hypergraph.

A directed hypergraph $G(V, E, <_V)$ is similar to an undirected hypergraph, with the addition of $<_V$ which is a partial order on the set of vertices. A topological ordering for G is a bijection $\vartheta : V \rightarrow \{1, 2, \ldots, \|V\|\}$ that preserves the partial order constraints. Namely, $\forall v_1, v_2$, if $v_1 <_V v_2$, then $\vartheta(v_1) < \vartheta(v_2)$. Definitions involving cutwidth for directed hypergraphs are the same as those for the undirected hypergraphs.

A combinational circuit $C(N, P)$ is represented as a set of nets/signals N and a set of gates P. Each gate $g \in P$ is represented as a set of ordered pairs of signals $\langle n_i, n_o \rangle$, where $n_i, n_o \in N$ and n_i is in the fanin of g and n_o is in the fanout of g. A combinational circuit $C(N, P)$ without feedback loops determines a directed hypergraph $G(V, E, <_V)$, where V is identical to the set N; each $e \in E$ corresponds to a set that consists of a signal v and all immediate fanouts V_O of v, i.e. V_O are the output signals of gates having v as an input; $<_V$ is a partial order of signals where $v_i <_V v_o$ iff $\langle v_i, v_o \rangle \in P$. We call ϑ a topological ordering of a circuit $C(N, P)$ if ϑ is a topological ordering for the corresponding direct hypergraph. The set of hyperedges E for a circuit can be formally defined as

$$E = \{\{v_i\} \cup V_O \mid v_i \in V \wedge \forall\, v \in V\ (v \in V_O \leftrightarrow \langle v_i, v \rangle \in P)\}$$

Definition 2.3 Combinational Circuit Cutwidth
Given a combinational circuit $C(N, P)$ and a topological ordering ϑ, the cutwidth of the circuit C under ϑ, $c(C, \vartheta)$, is the cutwidth of the corresponding directed hypergraph.

A sequential circuit is represented as $S(N, P, L)$, where N is a set of nets/signals, P is a set of gates, and L is a set of latches. Each gate or latch is represented as a set of ordered pairs of signals. For a sequential circuit $S(N, P, L)$, we represent a single time frame expansion of S as $C_S(N, P)$. In the single time frame expansion, the set of latches is removed, the input signals of the latches become pseudo-primary outputs, and the output signals of the latches become pseudo-primary inputs. We assume that there are no combinational feedback loops in a sequential circuit.

Definition 2.4 Sequential Circuit Cutwidth
Given a sequential circuit $S(N, P, L)$, let $C_S(N, L)$ be a single time frame expansion of S. An ordering function ϑ is valid for S, if it is a valid topological ordering for C_S and all the pseudo-primary inputs are ordered first and all the pseudo-primary outputs are ordered last in ϑ. The cutwidth of the circuit S under a valid ordering ϑ is the cutwidth of C_S under ϑ.

Typically symbolic simulation traverses a sequential circuit time frame by time frame. As a consequence, once generated, BDDs of the input signals of latches are live until one time frame is completely finished, and all BDDs of output signals of latches are live at the beginning of a new time frame. We will show that cutwidth of a circuit corresponds to the number of peak live BDDs in a symbolic traversal. We restrict the ordering for pseudo-primary inputs and pseudo-primary outputs to accurately model the actual circuit traversal in symbolic simulation.

3 Ordering Algorithms

In this section, we give algorithms to generate orderings that reduce cutwidth for both undirected and directed hypergraphs.

3.1 Undirected Hypergraph

It is well known that the following decision problem, called *minimum cut linear arrangement*, is NP-complete: For a given graph G and an integer k, does there exists an ordering function ϑ, so that the cutwidth $c(G, \vartheta)$, as defined in Section 2, is less than k? In this paper, we have designed an approximation algorithm which in practice could find orderings that reduce cutwidth for a given undirected hypergraph. Our algorithm is based on the divide-and-conquer paradigm. For each recursive bipartition, we have used an existing hypergraph partitioning package *hMetis* [7]. We have also implemented a simple version of the *terminal propagation* technique [5], which estimates the positions of connections between a subgraph and the remainder of the hypergraph, so that a partition with smaller crossing edges can be generated.

3.2 Directed Hypergraph

For directed hypergraphs, we have designed a similar divide-and-conquer algorithm based on *hMetis* to generate topological orderings that reduce cutwidth. Two subgraphs are called *directed partitions* of a hypergraph, if they are bipartitions of the hypergraph, and it is never the case that a vertex is in partition 1, but one of its fanout vertices is in partition 0. We take advantage of the fact that for sequential circuits, the pseudo-primary inputs are ordered first, and pseudo-primary outputs are ordered last. These are given to hMetis as constraints to help generate directed partitions. After a hypergraph is partitioned by hMetis, the produced bipartitions are generally not directed partitions. We refine the two partitions as follows: for each edge e' and each pair of vertices u and v in e', if u is in partition 1 and v is in partition 0 and $u <_V v$, then u is moved to partition 0. This process is repeated, until no violation of partial order constraints of vertices exists.

4 Cutwidth Based Heuristics

4.1 Symbolic Simulation

In symbolic simulation, BDDs for latches are computed in terms of primary inputs. We say a gate is *ready* when the BDDs for all its input gates have already been built. We say a gate is *dead* if the BDDs for all its output gates have already been built. To reduce the memory requirement, the BDD of a gate is deallocated once it becomes dead. Therefore, a topological order is used to schedule gates for building BDDs. For each gate, this ordering determines the lifetime of its BDD. For the whole circuit, it determines the maximal number of simultaneous live BDDs during simulation. We have the following key observation:

Observation 4.1 Given a sequential circuit, the number of peak live BDDs during symbolic simulation is the same as the cutwidth of the circuit.

The standard way of scheduling gates in symbolic simulation is to use either a queue, which corresponds to a BFS ordering, or a stack, which corresponds to a DFS ordering. We call this *BFS/DFS-based symbolic simulation*. The main drawback of using BFS/DFS ordering is that it is possible to accumulate many live BDDs before they become dead and are deleted. This will result in more intermediate BDD nodes and hence require more memory. In this paper, we use the algorithm described in Section 3.2 to generate a topological ordering of the circuit, which reduces cutwidth and consequently the number of peak live BDDs. In order to directly reduce the peak number of live BDD nodes, we use weighted directed hypergraph to represent a circuit, where the weight of a hyperedge approximates the BDD size of its corresponding source signal(recall that a hyperedge contains a source signal and its fanouts). In time frame based symbolic simulation, we use the BDD size of a signal in previous time frames as an estimate for the current time frame.

We call this algorithm *cutwidth-based symbolic simulation* (CUT_sim).

4.2 SAT Procedure

In this section, we combine cutwidth based heuristics with Davis-Putman-Logemann-Loveland (DPLL) style satisfiability algorithms. Several existing SAT solvers are based on extensions of DPLL algorithms. Examples include GRASP, SATO and CHAFF. GRASP [13] extends the basic DPLL algorithm by using conflict clauses to do caching and shows that caching significantly improves efficiency. In this paper, we have modified a version of GRASP to incorporate cutwidth based heuristics. Propositional formulas are represented in clause form (conjunctive normal form). The algorithms outlined in the Section 3.1 are used to generate orderings that reduce cutwidth for CNF formulas. We show that such orderings can be used for the variable splitting decision and improve caching of the SAT algorithms.

Figure 1 contains the modified algorithm that uses cutwidth based heuristics. For the ease of illustration, we describe a non-deterministic algorithm. In practice, nondeterministism is typically implemented using backtracking.

// V is the set of unassigned variables
// C is the set of clauses
// ϑ is the min_cutwidth variable ordering
1.**Algorithm** Search()
2. **if** $V == \emptyset$ **return** SATISFIABLE
3. $v \leftarrow$ PickElement(ϑ,V)
4. $v \leftarrow true/false$
5. **if** UnitPropogation() == CONFLICT
6. $c \leftarrow$ ConflictClauseCrossCut(v)
7. $C \leftarrow c \cup C$
8. **else**
9. Search()

Figure 1. Cutwidth-based DPLL.

In Line 3, PickElement(ϑ,V) returns the first unassigned variable in the ordering ϑ. In Line 4, variable v is nondeterministicly assigned a value of true or false. In practice, it can be implemented by backtracking. If an assignment to variable v leads to a contradiction, it will be detected in Line 5. In Line 6, ConflictClauseCrossCut(v) returns a conflict clause that contains only variables in the cutset(v), which is implemented by a backward traversal of the conflict graph. We have proven that such a conflict clause always exists. However, we omit the proof in this paper. This strategy is essential to establish the single exponential complexity with respect to cutwidth in our algorithm.

Theorem 4.1 *Given a set of clauses S and a variable ordering ϑ, assume that each clause in S contains at most a constant number of variables, the time complexity of cutwidth-based DPLL is $O(n.2^w)$, where n is $|S|$ and w is the cutwidth of S with respect to ϑ.*

5 Experimental Results

For all the results reported here, we use a 360M Hz Sun 6500 Enterprise server. We use the geometric mean to compute the average improvements in the algorithms discussed in Section 4. The improvement for our symbolic simulation algorithm is much bigger than the improvement for our SAT algorithm.

5.1 Results for SAT

The cutwidth-based DPLL algorithm in Section 4.2 is implemented on top of fgrasp [13]. Our new algorithm is referred to as CUT_SAT. Since the performance of a SAT solver can be influenced by the options used, we adopt the options from the SAT_Ex site [14], which are ‘‘+B2147483647 +C2147483647 +S2147483647 +g20 +rt4 +V0”. We compare our algorithm with the dynamic decision heuristic *DLIS*, which selects the decision variable, so that it satisfies the maximum number of clauses. The default timelimit is 10,000 seconds for both versions of the algorithm. Table 1 summarizes the our results for the DIMCAS benchmarks and the “dlx” *Superscalar Suite 1.0a* benchmarks [15]. In the table, FP represents fgrasp and CS represents CUT_SAT. The first column is the name of the benchmark class, the second column is the number of instances within that class. The third and fourth columns are the number of instances within each class that are finished by fgrasp and CUT_SAT within the timelimit. The fifth, sixth and seventh columns are the time used by fgrasp, CUT_SAT and the time CUT_SAT uses to derive an ordering. Note that only those benchmarks that are completed within the timelimit are counted. The last column is the average cutwidth for each benchmark class.

The benchmarks in Table 1 are organized into three categories: *trivial*, *difficult*, and *hard*.

- For the *trivial* benchmarks, both CUT_SAT and fgrasp run pretty fast, and there is no advantage using CUT_SAT.
- CUT_SAT is able to finish more benchmark instances than fgrasp within the timelimit for most of the *difficult* benchmarks. Our algorithm uses much less time to finish all the instances for the “h” and “par16” classes of examples from *DIMACS*. But for “ii32”, fgrasp is able to finish one more benchmark. Another interesting observation is that the performance of CUT_SAT

class	#M	#Finished		Time(sec)			Cutwidth
		FP	CS	FP	CS	Order	
aim-100	24	24	24	0.8	0.7	8.8	115
aim-200	24	24	24	8.9	4.9	20.3	230
aim-50	24	24	24	0.4	0.4	3.6	63
bf	4	4	4	2.1	2.6	64.8	200
dubois	12	12	12	2.7	0.2	17.5	13
ii8	14	14	14	5.4	1.6	73	398
jnh	50	50	50	6.7	41.5	49.2	648
pret	8	8	8	4.5	0.7	2.2	28
ssa	8	8	8	3.7	86.5	538	254
h	7	4	7	1828.7	377.7	209.8	94
ii16	10	9	10	140.7	672.9	466.9	1424
ii32	17	17	16	5.3	32.7	121.8	1144
par16	10	8	10	7329.9	1153.7	86.6	146
par32	10	0	0	0	0	0	311
f	3	0	0	0	0	0	2686
g	4	0	0	0	0	0	4720
dlx	9	2	4	1916.0	1461.2	4302.7	5311

Table 1. Comparison for fgrasp (FP) and CUT_SAT (CS)

can be predicted by the cutwidth of the CNF formula. For example, the average cutwidth for "h" and "par16" is approximately a hundred, but the single CNF formula within "ii32" that CUT_SAT fails to finish has a cutwidth of 2354. This is at least twice as big as the cutwidth for the rest of the instances within "ii32".

- Neither CUT_SAT nor fgrasp does very well on the *hard* class of benchmarks, but for the "dlx" from *Superscalar Suite 1.0a*, CUT_SAT could finish two more benchmarks, although this class of benchmarks has a very big average cutwidth.

Based on this table, our cutwidth-based DPLL algorithm gives significant improvements over the dynamic decision heuristic for SAT instances with cutwidth below several hundred. This suggests a conservative way to use the cutwidth-based algorithm. First, generate an ordering using the algorithm in Section 3, then use the cutwidth-based heuristics if the cutwidth of the ordering is small. As demonstrated by the "dlx" examples, for practical SAT instances in verification, the cutwidth-based algorithm could give better results even if the SAT instances have large cutwidth. Thus, it is possible to incorporate this algorithm for solving benchmarks with large cutwidth by alternating between different decision making heuristics.

5.2 Results for Symbolic Simulation

We have implemented the BFS, DFS and cutwidth-based (CUT_sim) symbolic simulation algorithms using CUDD 2.3.0. Our experiments are based on the 31 nontrivial circuits in the *ISCAS93* benchmark set. As far as the cutwidth between different orderings are concerned, on average DFS is 1.6 times smaller than BFS, and CUT_sim is 3.2 times smaller than DFS. The ordering time used by CUT_sim is usually small. The longest is 793 seconds for "s38417" which has about 24K gates.

We then carry out symbolic simulation experiments using these three ordering algorithms to verify the claim that an ordering with smaller cutwidth usually exhibits better performance with symbolic simulation. We have performed two sets of experiments. The first set of experiments described in Table 5.2 uses dynamic BDD variable reordering but does not use any pre-generated initial BDD variable ordering. The second set of experiments (which is omitted here because the results are similar to the first set of experiments) does not use BDD variable reordering, instead it uses the three BDD variable ordering files saved in the first set of experiments. We have omitted the results for BFS, which is slightly worse than DFS. Also in both sets of experiments, we start with all latches having value 0 in the first time frame. We symbolicly simulate each circuit up to 100 time frames or up to 20,000 seconds. In Tables 5.2, the first column is the name of the benchmark. The second and third columns are the numbers of symbolic simulation steps finished by DFS and CUT_sim. The fourth and fifth columns are the time (including cutwidth ordering time) used. The last two columns are the peak number of live BDD nodes in millions during simulation. Based on the second and third columns, CUT_sim finishes more simulation steps for 22 out of 31 circuits. In order to accurately compare the performance of the two algorithms, the time and BDD node-counts reported are collected when both DFS and CUT_sim complete *the same common maximal number of simulation steps*. The results are arranged into four categories according to runtime. In the the first category, DFS performs better than CUT_sim, but the differences are usually small. In the second category, CUT_sim is slightly better than DFS. In the third category, CUT_sim improves the run time by a factor between 2 and 9. In the last category, CUT_sim is more than an order of magnitude faster than DFS.

Finally, we summarize the results for BFS, DFS and CUT_sim. In terms of number of simulation steps finished and the time used, CUT_sim wins 27 cases, DFS wins 2 cases and BFS wins 2 cases. The average reduction of CUT_sim over BFS is 4.5 in time and 2.6 in space. The average reduction of CUT_sim over DFS is 3.9 in time and 2.4 in space.

6 Conclusion

This paper gives algorithms for computing variable orderings that reduce cutwidth. These algorithms work on both CNFs and on circuits. We show that the ordering for a circuit can be used to guide circuit traversal in symbolic simulation. The new traversal order reduces the number of live BDDs nodes better than either BFS or DFS order. We also show how a DPLL based SAT algorithm can be modi-

circuit	Finished steps		Time (minutes)		BDD nodes (M)	
	DFS	CUT	DFS	CUT	DFS	CUT
s35932	12	12	83.2	105.7	2.9	3.0
s526	71	70	230.9	143.8	5.3	4.3
s1269	4	4	3.7	2.2	0.4	0.4
s13207	87	88	312.2	290.1	3.1	2.9
s15850	48	49	296.1	279.3	8.5	7.1
s298	69	73	311.9	186.9	3.8	2.7
s382	68	68	153.2	152.5	5.2	5.0
s4863	5	5	24.4	12.5	0.9	0.6
s510	64	64	51.2	32.9	0.2	0.1
s713	16	18	281.7	225.3	3.8	3.5
s820	15	17	41.1	37.1	0.4	0.7
s832	16	17	168.5	126.8	0.7	1.0
s13207.1	36	37	266.6	129.2	5.7	3.3
s1423	11	13	37.8	18.7	1.8	0.7
s1494	15	17	41.9	10.3	0.5	0.4
s3271	15	24	128.3	36.1	3.6	1.1
s3330	6	6	108.6	52.5	2.5	1.9
s38417	17	19	215.7	67.7	3.7	1.4
s38584.1	12	13	162.6	59.6	3.3	2.0
s38584	19	19	151.6	67.7	6.0	6.2
s444	63	69	165.2	46.4	4.6	1.5
s641	16	17	304.2	149.1	4.2	3.1
s6669	3	3	81.0	22.2	2.7	1.1
s1512	14	16	248.7	26.0	5.2	0.7
s526n	64	70	169.3	31.9	4.0	0.9
s5378	17	20	228.8	30.4	4.3	0.5
s1488	16	100	103.5	0.07	1.1	5e-6
s386	17	19	149.4	1.0	6e-5	4e-5
s9234.1	13	14	73.5	7.3	1.9	0.3
s9234	23	42	174.3	1.2	7.5	8e-5
s953	12	13	98.7	3.7	2.1	0.3

Table 2. Comparison for DFS and CUT_sim

fied to use this ordering for variable splitting and conflict caching. The modified algorithm has single exponential complexity with respect to the cutwidth of the input clauses. Our experimental results demonstrate the effectiveness of cutwidth based heuristics for both symbolic simulation and SAT.

In the future we would like to explore methods to combine cutwidth based heuristics with the dynamic decision making in SAT algorithms. We would like to apply our cutwidth based algorithm to image computation with conjunctive partitioning [4]. Finally we would also like to try our cutwidth-based heuristics on other efficient SAT procedures and further improve the efficiency of the ordering algorithms.

References

[1] F. Aloul, I. Markov, and K. Sakallah. Faster SAT and Smaller BDDs via Common Function Structure. In *Proc. of the International Conference on Computer Aided Design (ICCAD)*, 2001.

[2] C.Leonard Berman. Circuit Width, Register Allocation, and Ordered Binary Decision Diagrams. *IEEE Transactions on Computer-Aided Design 4(1)*, pages 1059–1066, August 1991.

[3] R. E. Bryant. Symbolic Simulation–Techniques and Applications. In *27th Design Automation Conference*, pages 517–521, June 1990.

[4] Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

[5] A.E. Dunlop and B.W. Kernighan. A Procedure for Placment of Standard Cell VLSI Circuits. *IEEE Transactions on Computer-Aided Design 4(1)*, pages 92–98, 1985.

[6] David FERNANDEZ-BACA. Nonserial Dynamic Programming Formulations of Satisfiability. *Information Processing Letters 27*, pages 323–326, May 1988.

[7] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel Hypergraph Partitioning: Applications in VLSI Domain. In *34th ACM/IEEE Design Automation Conference*, 1997.

[8] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, Computer Science Department, 1992.

[9] In-Ho Moon and Fabio Somenzi. Border-Block Triangular Form and Conjunction Schedule in Image Computation. In *Proceedings of the Formal Methods in Computer Aided Design (FMCAD)*, November 2000.

[10] Matthew Moskewicz, Conor Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *38th ACM/IEEE Design Automation Conference(DAC)*, June 2001.

[11] David A. Plaisted, Armin Biere, and Yunshan Zhu. A Satisfiability Tester for Quantified Boolean Formulae. Submitted.

[12] Mukul Prasad, Philip Chong, and Kurt Keutzer. Why is ATPG easy? In *36th ACM/IEEE Design Automation Conference(DAC)*, pages 22–28, October 1999.

[13] J.P. Marques Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of the IEEE international Conference on Computer Aided Design (ICCAD)*, pages 220–227, 1996.

[14] Laurent Simon. The SAT_Ex Site. *http://www.lri.fr/ simon/satex/satex.php3*.

[15] M.N. Velev. Superscalar Suite 1.0a. *Available from: http://www.ece.cmu.edu/ mvelev*.