Variable Ordering for Efficient SAT Search by Analyzing Constraint-Variable Dependencies

Vijay Durairaj and Priyank Kalla

Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT-84112
{durairaj, kalla}@ece.utah.edu

Abstract: *This paper presents a new technique to derive an initial static variable ordering for efficient SAT search. Our approach not only exploits variable activity and connectivity information simultaneously, but it also analyzes how tightly the variables are related to each other. For this purpose, a new metric is proposed - the degree of correlation among pairs of variables. Variable activity and correlation information is modeled (implicitly) as a weighted graph. A topological analysis of this graph generates an order for SAT search. An algorithm called ACCORD (ACtivity - CORrelation - ORDering) is proposed for this purpose.*

While ACCORD rigorously analyzes constraint-variable dependencies, it does not account for the effect of decision-assignments on clause-variable dependencies. This issue motivates further refinements to our approach using literal activity and correlation measures - giving rise to the L'ACCORD algorithm. Using efficient implementations of the above, experiments are conducted over a wide range of benchmarks. The results demonstrate that: (i) the variable order generated by our approach significantly improves the performance of SAT solvers; (ii) time to derive this order is a fraction of the overall solving time. As a result, our approach delivers faster performance (often, by orders of magnitude) as compared to contemporary approaches.

1 Introduction

Contemporary SAT solvers have matured over the years and come a long way from the classical search procedures of Davis-Putnam (DP) [1] and Davis-Logemann-Loveland (DLL) [2]. Recent approaches [3] [4] [5] etc., employ sophisticated methods such as constraint propagation and simplification, conflict analysis, learning and non-chronological backtracks [3] [4] [5] to efficiently analyze and prune the search space. In recent past, a lot of effort has been invested in trying to understand the nature of the SAT problem. The works that deserve mention relate to symmetry analysis [6] [7], local search strategies [8], complexity of SAT *viz-a-viz* ATPG [9], relationship of BDD variable orderings and CNF search procedures [10], amount of search space analyzed [11], UNSAT core extraction [12] [13], among others [14] [15] [16].

An important aspect of CNF-SAT is to derive an ordering of variables to guide the search. The order in which variables (and correspondingly, constraints) are resolved significantly impacts the performance of SAT search procedures. Boolean functions arising in many applications represent some spatial, casual or logical dependencies (or connections) among variables. Therefore, analysis of these clause-variable dependencies provides useful information that can be exploited to guide CNF-SAT search procedures. **Variable activity** and **clause connectivity** are often considered as qualitative and quantitative metrics to model clause-variable dependencies. Activity of a variable (or literal) is defined as the number of its occurrence among all the clauses of a given SAT problem [17]. Most conventional SAT solvers [4] [5] [3] employ variable/literal-activity based branching heuristics to resolve the constraints.

Connectivity of constraints has also been used as a heuristic approach to derive variable orderings for SAT search. Loosely speaking, two clauses are said to be "connected" if one or more variables are common to their support. Clause connectivity can be modeled by representing CNF-SAT constraints as (hyper-) graphs and, subsequently, analyzing the graph's topological structure. Tree decomposition techniques have been proposed in literature [18] [19] for analyzing connectivity of constraints in constraint satisfaction programs (CSP). Such techniques identify decompositions with **minimum tree-width**, thus enabling a partitioning of the overall problem into a chain of connected constraints. Recently, such approaches have also found application in those problems that can be modeled as DPLL-based CNF-SAT search [19] [20] [21] [22] [23] [24]. Various approaches operate on such partitioned tree structures by deriving an order in which the partitioned set of constraints are resolved [20] [21] [22] [24]. MINCE [10] employs CAPO placer's mechanism [25] to find a variable order such that the clauses are resolved according to their chain of connectivity. Bjesse *et. al.* [21] proposed tree decomposition based approaches to guide variable selection and conflict clause generation. Aloul *et. al.* have proposed a fast, heuristic based approach, FORCE [26], as an alternative to the computationally complex approach of MINCE. Recently, Durairaj *et. al.* [27] [28] [29] proposed hypergraph bi-partitioning based constraint decomposition scheme that employs both variable activity and clause connectivity simultaneously to derive a variable order.

This paper presents a new approach to derive an initial static ordering for SAT search by rigorously analyzing constraint-variable dependencies. Experimental results demonstrate that our approach is faster and more robust than the contemporary variable ordering techniques, and it improves the performance of SAT solvers (in many cases by orders of magnitude).

The paper is organized as follows. The following Section analyzes the limitation of previous work. Section 3 outlines the specific contributions of this research. The variable order generation approaches are presented in Sections 4 and 5, followed by experimental results and analysis in Section 6. Finally, Section 7 concludes the paper.

2 Limitations of Previous Work

The computational complexity (exponential) of minimum tree-width decomposition algorithms results in large compute times to search for the variable order. The technique of Dechter et. al. [18] is shown to be time exponential in the tree-width. Algorithms for approximating tree-width with bounded error [23] are also shown to be too costly for industrial problems [21]. As a result, these techniques are impractical for large/hard CNF-SAT problems.

In general, the time required to derive a variable order should be small as compared to the subsequent SAT solving time - it should certainly not exceed the solving time. Unfortunately, for large and hard SAT problems, it has been observed that MINCE [10] [30] and Amir's tool [20] [31] require unacceptably long time just to derive the variable order. This behaviour is depicted in Table 1, which compares the time required to derive the variable order by Amir's tool [31], MINCE [30], FORCE [26] and the hypergraph partitioning based technique (HGPart) of [27] against zCHAFF's (2003 version) *solve time*. It can be observed from the table that in order to derive the variable order, both Amir's and MINCE approach suffer from long compute times - *much longer than the default SAT solving time*. In contrast, FORCE and HGPart can derive the variable order much faster than the other two. This clearly demonstrates the computational limitations of Amir's and MINCE approach; as such they are too expensive to be applicable for practical SAT problems. While, FORCE is fast, the quality of the variable order does not consistently improve the performance of SAT solvers.

Table 1. Comparison of original zCHAFF runtime, Amir's, MINCE's, FORCE's and HGPart's partitioning time

Benchmark	Vars/ Clauses	zCHAFF'03 (sec)	Amir (sec)	MINCE (sec)	FORCE (sec)	HGPart (sec)
c2670	2.5K/6.4K	1.48	16.54	8.23	0.94	1.15
c3540	3.4K/9.2K	20.57	23.66	13.95	0.83	1.42
c5315	5.0K/14.1K	34.62	54.56	25.51	6.52	1.25
c7552	5.5K/15.1K	105.97	71.30	24.43	3.81	1.76
4pipe	5.2K/80.2K	129	505.83	93.1	1.36	13.84
5pipe	9.5K/195K	196.53	>2000	267.2	2.82	38.40

Table 2. Quality of variable order derived by FORCE

Benchmark	Vars/ Clauses	zCHAFF'03 (sec)	FORCE (sec)
c3540	3431/9262	**20.57**	36.55
c5315	4992/14151	34.62	**9.25**
c7552	5466/15150	**105.97**	136.55
4pipe_q0_k	5380/69072	**66.71**	87.78
5pipe_q0_k	10026/154409	649.9	**375.37**
clus_set1	1200/4800	**59.62**	>1000

Table 2 presents some results that reflect the inconsistency in the quality of the variable order derived by FORCE. Since MINCE and Amir's tool have been shown to be impractical (too slow), we analyze the variable order derived by FORCE. The order derived by FORCE was given to the zCHAFF solver (2003 version) to resolve constraints - the runtime is given in Column 4. Column 3 corresponds to original zCHAFF runtime (VSIDS heuristic). From the results, it can be observed that the the variable order generated by FORCE does not consistently enhance the performance of the SAT solver[1]. In fact, for some benchmarks, it degrades the SAT solver performance by an unacceptably long time.

Recent approach of [27] has shown to deliver good results. Not only can the variable order be derived in a reasonable amount of time, the quality of the order also consistently improves the performance of the solver. However, as this technique relies on hypergraph partitioning, there is no direct control over the decomposition. As a result, many problems (such as the NQueens) cannot be decomposed efficiently. Another limitation of these techniques is that while they do analyze variable activity, clause connectivity or both, however, they do not analyze how tightly the variables are connected to each other. As a result, tightly connected, hard problems may not realize the run-time improvements.

3 Research Contributions

As discussed earlier, efficient CNF-SAT decision heuristics should analyze both variable activity and connectivity simultaneously for faster constraint resolution. Moreover, they should also analyze how tightly the variable are related to each other. Furthermore, the procedure to generate such an order should be fast, scalable and robust enough to handle large SAT problems. This paper proposes an efficient variable order generation procedure that attempts to fulfill the above criteria/requirements.

The contributions of our work are as follows:

1. We present an efficient technique that analyzes constraint-variable dependencies to derive an ordering of variables to guide SAT diagnosis. Both variable activity and connectivity information is exploited to derive the order.
2. In order to analyze how tightly two or more constraints are related (in terms of common variables), we propose a new metric called **the degree of correlation** among pairs of variables.
3. Variable activity and correlation information is (implicitly) modeled as a weighted graph, the topological analysis of which enables the derivation of such an order. An efficient algorithm (ACCORD) is described for this purpose.
4. In order to analyze the effect of decision-assignments made by SAT solver on the variable ordering, further refinements to the ACCORD algorithm are proposed.

[1] Mince also exhibits similar phenomenon.

5. Experiments conducted over a large and varied set of benchmarks demonstrate that our approach is fast, robust and scalable; quality of the variable order is reflected in the impressive speed-up achieved for SAT solving. Particularly for hard instances, orders of magnitude speed-up is achieved in many cases.
6. The variable order generation time is small as compared to the overall solving time.

4 Variable Order Computation: Analyzing Constraint-Variable Dependencies

It is our desire to derive a variable order for SAT search by analyzing clause-variable relationships. The importance of branching on high activity variables is well understood [17]. Analyzing the connectivity of constraints is also important for constraint resolution. Contemporary techniques address both of the above issues. However, 'how tightly are the variables related?' - this feature too should not be overlooked. For this purpose, we propose a metric that measures how tightly the variables are connected/related. We define this metric as follows:

Definition 41 *Two variables x_i and x_j are said to be* **correlated** *if they appear together (as literals) in one or more clauses. The number of clauses in which the pair of variables (x_i, x_j) appear together is termed as their* **degree of correlation**.

4.1 Problem Modeling

In our approach, the constraint-variable relationship of a given CNF-SAT problem is modeled as a weighted graph. The variables (as opposed to literals) form the vertices, while edges denote the correlation/connectivity between them. Associated with each variable is its activity, which is modeled as an integer value within the node. The edge weights represent the degree of correlation between the variables/vertices. For example, if two variables x_i, x_j appear together in n clauses, then the weight of the edge e_{ij} connecting them is n.

An ordering of the nodes (variable order) can be performed by analyzing the graph's topology. We now describe our approach by means of an example corresponding to the CNF-SAT problem shown in Fig. 1. Its corresponding weighted graph is depicted in Fig. 2(a).

4.2 ACCORD: Activity-Correlation based Ordering

We begin the search by first selecting the highest active variable, i.e. the node that has the highest internal weight. The variable is marked and the node is added to a set called *supernode*. The variable is also stored in a list (*var_ord_list*). The SAT tool should branch on this variable first. It can be observed from the graph (weight within the nodes) that the activity of variables $\{e, u, g, v\}$ is the

$$(a + b + \overline{u})$$
$$(\overline{a} + u)\,(\overline{b} + u)$$
$$(u + \overline{g})\,(c + \overline{g})$$
$$(\overline{u} + \overline{c} + g)$$
$$(a + \overline{e})\,(\overline{b} + \overline{e})$$
$$(\overline{a} + b + e)$$
$$(v + d + \overline{w})$$
$$(\overline{v} + w)\,(\overline{d} + w)$$
$$(e + g + \overline{v})$$
$$(\overline{e} + v)\,(\overline{g} + v)$$

Fig. 1. An example CNF-SAT problem

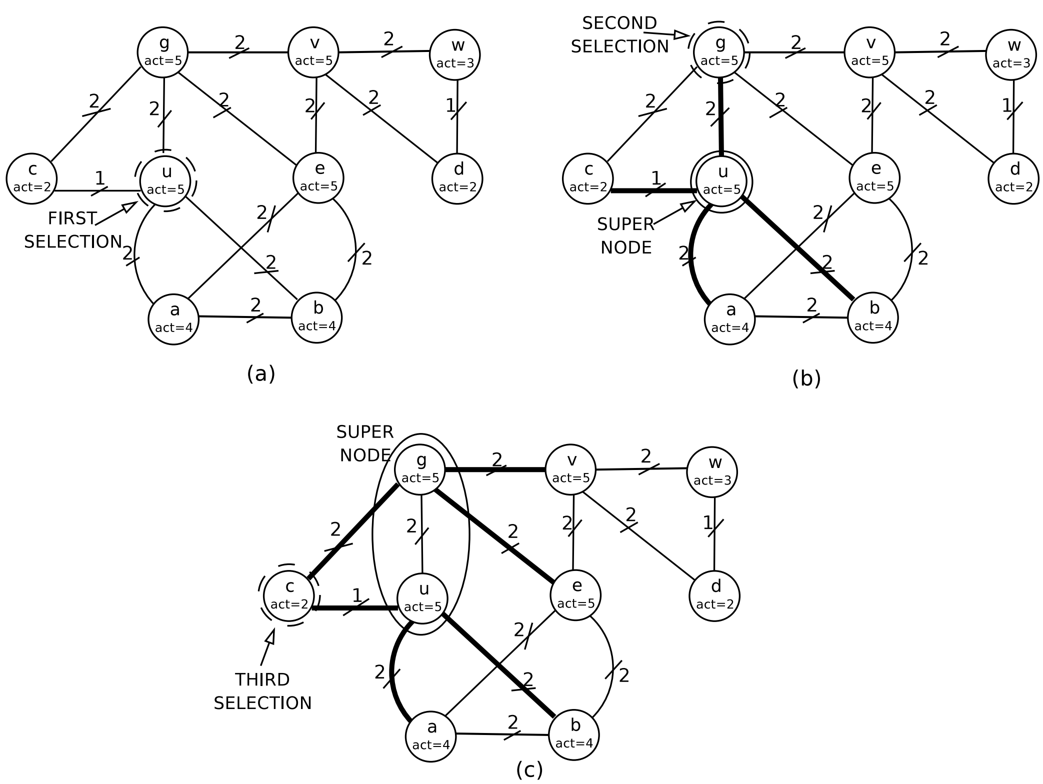

Fig. 2. Weighted Graph - Edge weights denote the degree of correlation between the variables (vertices). Variable activity depicted within the node.

highest (= 5). We select one of these variables (randomly or lexicographically) as the initial branching variable. Let us select the variable u, shown in Fig. 2(a) as "First Selection." The variable u becomes the supernode.

Now we need to identify the set of variables connected to this supernode. All the nodes that share at least one edge with the supernode are identified. The nodes $\{g, c, a, b\}$ are connected to the supernode u, as shown in Fig. 2(b). One of these variables $\{g, c, a, b\}$ is to be selected as the next branching variable. We consider the degree of correlation, modeled as edge weights, as the metric to identify this variable. The nodes that share an edge with the supernode are sorted in *decreasing order of the sum of their degrees of correlation with the nodes in the supernode.* In our example, variable c has the degree of correlation = 1. On the other hand, the variables $\{g, a, b\}$ have the same degree of correlation with node u, which is equal to 2; hence, we consider variables $\{g, a, b\}$. In order to break this tie, *we further distinguish these variables according to their activity.* Since the activity of variable g is higher than that of variables $\{a, b\}$, g is selected as the next branching variable. This is shown in Fig. 2(b) as "Second Selection." Moreover, the node g is added to the current supernode set; *supernode* = $\{u, g\}$. The variable order derived so far is *var_ord_list* = $\{u, g\}$.

As the next step, those variables are identified that share an edge with the supernode. These variables are $\{v, e, c, a, b\}$ as shown in Fig.2(c) with highlighted incident edges. Consider the nodes $\{v, e, a, b\}$. They share only one edge with the supernode with weights (correlation) = 2. On the other hand, node c has two edges incident on the supernode. Therefore, the correlation modeled by both edges should be accounted for. This is computed as the sum of the degrees of correlation between c and the supernode $\{g, u\}$ = 2+1 = 3. Since this sum is the highest for c, it is selected as the next branching variable. This is shown in Fig. 2(c) as third selection. The supernode set and the *var_ord_list* are correspondingly updated. The above procedure is repeated until all the nodes are ordered and included with in the supernode. The final derived variable order is $\{u, g, c, v, e, a, b, w, d\}$.

Our approach is inspired from the Prim's Minimum Spanning Tree (MST) algorithm [32]. We have named it the **ACCORD** (<u>AC</u>tivity <u>CO</u>rrelation <u>ORD</u>ering) algorithm. The Pseudo code for the ACCORD algorithm is presented in Algorithm 1. The Variable-Clause database, mentioned in lines 1-3, is implemented using the array of arrays data-structure available within contemporary SAT solvers [4] [33]. For example, the Variable-Clause database corresponding to the CNF-SAT problem of Fig 1 is shown in Fig. 3. The variable a appears in Clauses {1, 2, 7, 9}. Hence, the first element of Variable-Clause database array is an array containing {1, 2, 7, 9}. Similarly, the second element of the array is an array containing {1, 3, 8, 9} corresponding to variable b; and so on. The algorithm analyzes the clause-variable database, and for each variable, it computes its correlated variables. Using the activity and correlation measures, the variable ordering is computed.

Algorithm 1 Pseudo code for ACCORD

```
INPUT = CNF Clauses and Variable-Clause Database ;
/* Variable-Clause database is implemented as array of arrays */
/* For each variable x_i, the Variable-Clause database contains a list of clauses in which x_i appears */
OUTPUT = Variable order for SAT search ;
activity_list = Array containing activity of each variable ;
var_order_list = Initialize variable order according to activity ;
connectivity_list = Initialize to zero for all variables ;
int i = 0;
while i != number of variables do
  /* Implicitly, supernode = {var_order_list[0], ..., var_order_list[i]} */
  next_var = var_order_list[i] ;
  correlation_list = find variables connected to next_var using Variable-Clause database ;
  for all var ∈ correlation_list do
    connectivity_list[var]++ ; /* Compute correlation */
    adjust_variable_order(var) ;
    /* Linear sort is used to update the variable order corresponding to both connectivity_list as well as activity_list */
  end for
  i = i+1;
  /* At this point, {var_order_list[0], ..., var_order_list[i]} is the current variable order*/
end while
return(var_order_list) ;
```

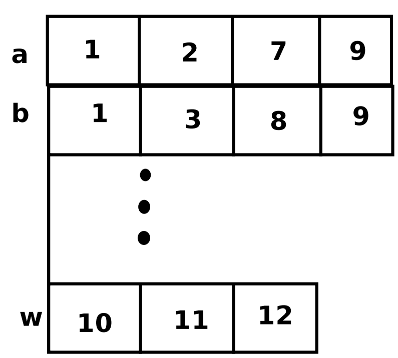

Fig. 3. Variable-Clause Database

4.3 Complexity of ACCORD

A weak upper bound for the ACCORD algorithm can be found by assuming that every variable appears in every other clause. With this assumption, an upper bound on the time complexity of ACCORD can be derived as $O(V \cdot (V \cdot C + V^2))$, where V represents the number of variables and C represents the number of clauses.

5 L' ACCORD: Overcoming the Limitations of ACCORD

Most conventional SAT solvers employ literal-activity based branching heuristics to resolve the constraints. Also, when the variable order generated by ACCORD is given to a SAT solver, the solver will branch on a literal corresponding to the variable selected by the ACCORD. Clauses corresponding to this literal will be satisfied due to this assignment. ACCORD does not analyze this effect of decision-assignments on the variable correlation. Consider the following set of clauses.

$$(x + a + e)(x + b + f)(x + c + \overline{e})$$
$$(\overline{x} + d + f)(\overline{x} + c)$$

Here, the literal x has the highest activity and the SAT solver will assign $x = 1$. This satisfies the first three clauses. The ACCORD algorithm will consider the variables {a, e, b, f, c, d} to compute their respective degrees of correlation with respect to x. However, due to the assignment $x = 1$, it should suffice to look at only those clauses in which $\overline{x}$ appears, as they are unsatisfied. This means that the correlated variables to $\overline{x}$ ({d, f, c}) and their corresponding constraints should be analyzed to derive a variable order. In order to exploit such a behaviour, we have implemented the above modifications to the ACCORD algorithm. Instead of analyzing variable-to-variable correlations, we now analyze correlations between a literal and its connected variables. The modified algorithm is now called L' ACCORD.

6 Experimental Results and Analysis

The ACCORD and L' ACCORD algorithms have been programmed within the zCHAFF [4] solver using its native data-structures. The algorithms analyze the constraint-variable relationships of the given CNF-SAT problem and derive a variable order for SAT search. Using this as the initial order, the SAT tool (zCHAFF) performs a search for the solutions. On encountering conflicts, we allow the solver to add conflict-induced clauses and proceed with its book-keeping and (non-chronological) backtracking procedures. In other words, ACCORD/L' ACCORD provide only an initial static ordering. zCHAFF's VSIDS heuristic updates this order dynamically, when conflict clauses are added. Hence, our approach is not a replacement for VSIDS; it is to be used in conjunction with it.

Using this setup, we have conducted experiments over a large set of benchmarks that include: i) Microprocessor verification benchmarks [34]; and ii) some of the hard instances specifically created for the SAT competition (all three categories - industrial, handmade and random). We conducted our experiments on a Linux workstation with a 2.6GHz Pentium-IV processor and 512MB RAM.

6.1 ACCORD versus L' ACCORD comparison

Table 3 compares the quality of orders generated by ACCORD and L' ACCORD. It is clear from the table that L' ACCORD generates a better quality variable order than ACCORD, always improving the SAT solver performance.

Table 3. Run-time Comparison of ACCORD and L' ACCORD

	zCHAFF	ACCORD + zCHAFF			L' ACCORD + zCHAFF		
Bench-mark	Solve (sec)	Var. Time(s)	Solve Time(s)	Total Time(s)	Var. Time(s)	Solve Time(s)	Total Time(s)
clus-2020-1	>1000	0.01	746.62	746.63	0.01	666.32	**666.33**
clus-1010-3	>1000	0.01	>1000	–	0.01	700.85	**700.86**
conn-n600-939	135.54	0.01	82.82	82.83	0	0.42	**0.42**
conn-n600-945	581.56	0.01	534.64	534.65	0.01	98.02	**98.03**
icos-stretch	102.99	0	111.39	111.39	0	78.56	**78.56**
marg-33-add8ch	247.73	0	21.8	21.8	0	20.01	**20.01**
marg-35-1452	>1000	0	>1000	–	0	289.17	**289.17**
mm-1x10-1488	654.16	0.01	>1000	–	0.01	237.48	**237.49**
mm-2x2-50-149	39.8	0.02	27.64	27.66	0.05	24.9	**24.95**
qwh-35-405	39.73	0.01	114.45	114.46	0.03	5.58	**5.61**
urqh1c3x3	284.48	0	8.53	**8.53**	0	57.96	57.96
urqh2x4	302.79	0	33.39	33.39	0	26.53	**26.53**
urqh2x5	>1000	0	>1000	–	0	612.96	**612.96**
urqh1c3x4	>1000	0	>1000	–	0	320.62	**320.62**
unif-r4-2	>1000	0	>1000	–	0	198.47	**198.47**
unif-r4-7	>1000	0	515.24	515.241	0	151.25	**151.25**
unif-r4-9	>1000	0	350.12	350.12	0	141.95	**141.95**
3bitadd_31	68.88	0.14	43.03	43.17	0.09	4.1	**4.19**
3bitadd_32	7.43	0.15	0.86	1.01	0.1	0.02	**0.12**
9dlx-vliw-iq1	504.47	12.79	464.85	477.64	15.69	381.09	**396.78**

6.2 L' ACCORD versus Contemporary Techniques

Table 4 presents some results that compares the quality of the variable order derived by L' ACCORD with those of zCHAFF (latest version developed in 2004) and hypergraph partitioning (HGPart) based order [27]. For a fair comparison, zCHAFF is used as the base SAT solver for all experiments - just the variable orders are different. As compared to zCHAFF and HGPart, the variable order generated by L' ACCORD results in a orders of magnitude speedup in SAT solving. Particularly for more difficult problems, our approach significantly outperforms the other two.

Table 4. Run-time Comparison of L' ACCORD with zCHAFF and Hypergraph Partitioning based Technique

			zCHAFF	HGPart + zCHAFF			L' ACCORD + zCHAFF		
Bench-mark	Vars/ Clauses	SAT/ UNSAT	Solve (sec)	Var. Time(s)	Solve Time(s)	Total Time(s)	Var. Time(s)	Solve Time(s)	Total Time(s)
3bitadd_31	8432/21210	S	68.88	3.023	0.55	**3.573**	0.09	4.1	4.19
3bitadd_32	8704/32316	S	7.43	3.417	0.67	4.087	0.1	0.02	**0.12**
clus-2020-1	1200/4800	S	>1000	1.421	>1000	–	0.01	666.32	**666.33**
clus-2020-2	1200/4800	S	688.19	1.81	9.12	**10.93**	0.04	37.17	37.21
clus-4530-2	1200/4800	S	975.83	1.846	43.85	**45.696**	0.02	630.55	630.57
clus-1010-3	1200/4919	S	>1000	1.701	704.66	706.361	0.01	700.85	**700.86**
color-10-3	300/6475	S	129.57	0.375	19.01	**19.385**	0	25.05	25.05
conn-n600-939	576/6864	S	135.54	1.196	28.96	30.156	0	0.42	**0.42**
conn-n600-945	596/7157	S	581.56	1.145	>1000	–	0.01	98.02	**98.03**
icos-stretch	45/352	U	102.99	0.199	93.84	94.039	0	78.56	**78.56**
marg-33-add8ch	41/272	U	247.73	0.071	142.53	142.601	0	20.01	**20.01**
marg-33-add8	41/224	U	3.44	0.074	1.36	1.434	0	1.29	**1.29**
marg-35-1452	61/280	U	>1000	0.085	>1000	–	0	289.17	**289.17**
mm-1x10-1488	1120/7220	S	654.16	1.057	>1000	–	0.01	237.48	**237.49**
mm-2x2-50-1496	60/32000	U	39.8	3.815	31.37	35.185	0.05	24.9	**24.95**
mm-2x3-66-1502	1698/48771	U	17.32	4.386	16.48	20.866	0.08	13.55	**13.63**
qwh-35-405	1597/10658	S	39.73	1.869	84.59	86.459	0.03	5.58	**5.61**
urqh1c3x3	41/204	U	284.48	0.078	8.92	**8.998**	0	57.96	57.96
urqh2x4	42/336	U	302.79	0.077	71.07	71.147	0	26.53	**26.53**
urqh2x5	53/432	U	>1000	0.117	>1000	–	0	612.96	**612.96**
urqh1c3x4	58/476	U	>1000	0.143	196.78	**196.923**	0	320.62	320.62
unif-r4-2	500/2000	S	>1000	0.447	>1000	–	0	198.47	**198.47**
unif-r4-5	500/2000	S	884.04	0.491	91.88	**92.371**	0	129.79	129.79
unif-r4-7	500/2000	S	>1000	0.451	582.53	582.981	0	151.25	**151.25**
unif-r4-9	500/2000	S	>1000	0.493	711.65	712.143	0	141.95	**141.95**
ferry10	2958/20791	S	3.54	3.549	0.73	4.279	0.04	0.07	**0.11**
ferry12	4222/32199	S	238.5	8.189	134.78	142.969	0.09	66.51	**66.6**
rotmul	5980/35229	U	174.7	5.888	153.26	**159.148**	0.28	159.53	**159.81**
9dlx-vliw-iq1	24604/261473	U	504.47	68.83	443.79	512.62	15.69	381.09	**396.78**

The table 5 depicts some results for the microprocessor pipeline verification benchmarks. These experiments were conducted using both 2003 and 2004 versions of the zCHAFF SAT solver. For both experiments, the same variable order derived by L' ACCORD is used. Note that, using the L' ACCORD's order, zCHAFF-2003 tool is able to improve the performance significantly. zCHAFF-2004 follows upon its earlier versions by implementing the efficient conflict analysis procedures. As a result, these benchmarks can be quickly solved by the 2004 version of the solver and the impact of L' ACCORD is minimal.

Table 5. Run-time Comparison of L' ACCORD with zCHAFF'03 and zCHAFF'04

		zCHAFF'03	L'ACCORD+zCHAFF'03			zCHAFF'04	L' ACCORD+zCHAFF'04	
Bench-mark	Vars/ Clauses	Solve (sec)	Var. Time(s)	Solve Time(s)	Total Time(s)	Solve Time(s)	Solve Time(s)	Total Time(s)
4pipe	5237/80213	129	0.56	56.1	56.66	8.02	11.96	12.52
5pipe	9471/195452	196.53	2.36	54.46	56.82	18.34	16.59	18.95
4pipe_k	5095/79489	208.81	0.6	40.85	41.45	8.34	14.53	15.13
5pipe_k	9330/189109	871.79	2.25	388.32	390.57	40.75	31.29	33.54
4pipe_q0	5380/69072	66.71	0.48	73.86	74.34	6.37	8.59	9.07
5pipe_q0	10026/154409	649.9	1.53	214.89	216.42	25.21	26.46	27.99
6pipe	15800/394739	>2000	6.53	827.06	833.59	132.82	144.72	151.25
6pipe_k	15346/408792	105.52	8.76	141.45	150.21	71.49	57.64	66.4
6pipe_q0	16775/315960	104.48	4.88	82.44	87.32	43.37	46.65	51.53
7pipe_q0	26512/536414	>2000	12.45	1523.16	1535.61	284.97	265.3	277.75
8pipe_q0	39434/887706	>2000	31.16	>2000	–	819.13	892.32	923.48

7 Conclusions and Future Work

This paper has advocated the need to analyze constraint-variable relationships to derive an ordering of variables to guide SAT diagnosis. To analyze the tightness of the connectivity between variables, we have a proposed the degree of correlation as a qualitative and quantitative metric. Our technique models the constraint variable dependencies on a weighted graph and analyzes the graph's topological structure to derive the order. Our approach is fast, robust, scalable and can handle a large set of variables and constraints. The variable order derived by our procedure improves the performance of the solver by one or more orders of magnitude. As part of future work, we are exploring a dynamic variable order update strategy to be employed when conflict clauses are added to the database.

References

1. M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory", *Journal of the ACM*, vol. 7, pp. 201–215, 1960.
2. M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving", *in Communications of the ACM, 5:394-397*, 1962.
3. J. Marques-Silva and K. A. Sakallah, "GRASP - A New Search Algorithm for Satisfiability", *in ICCAD'96*, pp. 220–227, Nov. 1996.

4. M. Moskewicz, C. Madigan, L. Zhao, and S. Malik, "CHAFF: Engineering and Efficient SAT Solver", *in In Proc. Design Automation Conference*, pp. 530–535, June 2001.
5. E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust Sat-Solver", *in DATE, pp 142-149*, 2002.
6. F. Aloul, A. Ramani, I. Markov, and K. Sakallah, "Solving Difficult SAT Instances in the Presence of Symmetry", *in Proc. DAC*, pp. 731–736, 2002.
7. J. Crawford, M. Ginsberg, E. M. Luks, and A. Roy, "Symmetry-Breaking Predicates for Search Problems", *in Proceedings of the Fifth International Conference on Knowledge Representation and Reasoning (KR '96)*, 1996.
8. B. Selman, H. Kautz, and B. Cohen, "Local Search Strategies for Satisfiability Testing", *DIMACS Series in Mathematics and Theoretical Computer Science*, vol. 26, pp. 521–532, 1996.
9. M. Prasad, P. Chong, and K. Keutzer, "Why is ATPG Easy?", *in Design automation Conf.*, pp. 22–28, June 1999.
10. F. Aloul, I. Markov, and K. Sakallah, "Mince: A static global variable-ordering for sat and bdd", *in International Workshop on Logic and Synthesis*. University of Michigan, June 2001.
11. F. Aloul and *et al.*, "Satometer: How Much Have We Searched?", *IEEE Trans. CAD*, vol. 22, pp. 995–1004, 2003.
12. L. Zhang and S. Mailk, "Extracting Small Unsatisfiable Cores from Unsatisfiable Boolean Formula", *in Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, 2003.
13. E. Goldberg and Y. Novikov, "Verification of Proofs of Unsatisfiability for CNF Formulas", *in Design, Automation and Test in Europe (DATE)*, 2003.
14. S. Pilarski and G. Hu, "SAT with Partial Clauses and Backleaps", *in In Proc. DAC*, pp. 743–746, 2002.
15. R. Damiano and J. Kukula, "Checking Satisfiability of a Conjunction of BDDs", *in DAC, pp 818-823*, 2003.
16. P. Kalla, Z. Zeng, M. J. Ciesielski, and C. Huang, "A BDD-based Satisfiability Infrastructure using the Unate Recursive Paradigm", *in Proc. Design Automation and Test in Europe, DATE'00*, 2000.
17. J. P. M. Silva, "The Impact of Branching Heuristics in Propositional Satisfiability Algorithms", *in Portuguese Conf. on Artificial Intelligence*, 1999.
18. R. Dechter and J. Pearl, "Network-based Heuristics for Constraint-Satisfaction Problems", *Artificial Intelligence*, vol. 34, pp. 1–38, 1987.
19. E. Amir and S. McIlraith, "Partition-Based Logical Reasoning", *in 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'2000)*, 2000.
20. E. Amir and S. McIlraith, "Solving Satisfiability using Decomposition and the Most Constrained Subproblem", *in LICS workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*, 2001.
21. P. Bjesse, J. Kukula, R. Damiano, T. Stanion, and Y. Zhu, "Guiding SAT Diagnosis with Tree Decompositions", *in Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, 2003.
22. A. Gupta, Z. Yang, P. Ashar, L. Zhang, and S. Malik, "Partition-based Decision Heuristics for Image Computation using SAT and BDDs", *in Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design (ICCAD)*, pp. 286–292. IEEE Press, 2001.
23. E. Amir, "Efficient Approximation for Triangulation of Minimum Treewidth", *in 17th Conference on Uncertainty in Artificial Intelligence (UAI '01)*, 2001.

24. J. Huang and A. Darwiche, "A structure-based variable ordering heuristic for SAT", *in Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1167–1172, August 2003.
25. A. Caldwell, A. Kahng, and I. Markov, "Improved Algorithms for Hypergraph Bipartitioning", *in Proc. Asia-Pacific DAC*, 2000.
26. F. A. Aloul, I. L. Markov, and K. A. Sakallah, "FORCE: A Fast and Easy-To-Implement Variable-Ordering Heuristic", *in Proceedings of the 13th ACM Great Lakes symposium on VLSI*, pp. 116–119, 2003.
27. V. Durairaj and P. Kalla, "Guiding CNF-SAT Search via Efficient Constraint Partitioning", *in ICCAD*, pp. 498 – 501, 2004.
28. V. Durairaj and P. Kalla, "Exploiting Hypergraph Partitioning for Efficient Boolean Satisfiability", *in HLDVT*, pp. 141–146, 2004.
29. V. Durairaj and P. Kalla, "Dynamic Analysis of Constraint-Variable Dependencies to Guide SAT Diagnosis", *in HLDVT*, pp. 135–141, 2004.
30. F. Aloul, I. Markov, and K. Sakallah, "MINCE", http://www.eecs.umich.edu/ faloul/Tools/mince/.
31. E. Amir and S. McIlraith, "The Partitioning and Reasoning Project", http://www-faculty.cs.uiuc.edu/ eyal/decomp/.
32. R. C. Prim, "Shortest connection networks and some generalizations", *The Bell System Technical Journal*, vol. 36, pp. 1389–1401, 1957.
33. N. Eén and N. Sörensson, "An Extensible SAT Solver", *in 6th International Conference, SAT*, 2003.
34. M. Velev, "Sat benchmarks for high-level microprocessor validation", http://www.cs.cmu.edu/ mvelev.