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. Also, the effect of decision-assignments on clause-variable dependencies is taken into account during this analysis. An algorithm called ACCORD (ACtivity - CORrelation - ORDering) is proposed for this purpose. 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 as compared to contemporary approaches.

1 Introduction

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. **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 [1]. Most conventional SAT solvers [2] [3] [4] 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 [5] [6] 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 [6] [7]. Various approaches operate on such partitioned tree structures by deriving an order in which the partitioned set of constraints are resolved [6] [7] [8] [9] [10] [7] [11] [10]. Recently, Durairaj *et. al.* [12] proposed hypergraph bi-partitioning based constraint decomposition scheme (HGPART) that employs both variable activity and clause connectivity simultaneously to derive a variable order.

The above connectivity/tree-decomposition/partitioning-based methods that guide SAT diagnosis have one or more of the following limitations: (i) they suffer from large compute times to search the variable order [5] [9] [6]; (ii) the quality of the variable order does not consistently improve the performance of SAT solvers [9] [11]; (iii) there is no direct control over the decomposition [12]. 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.

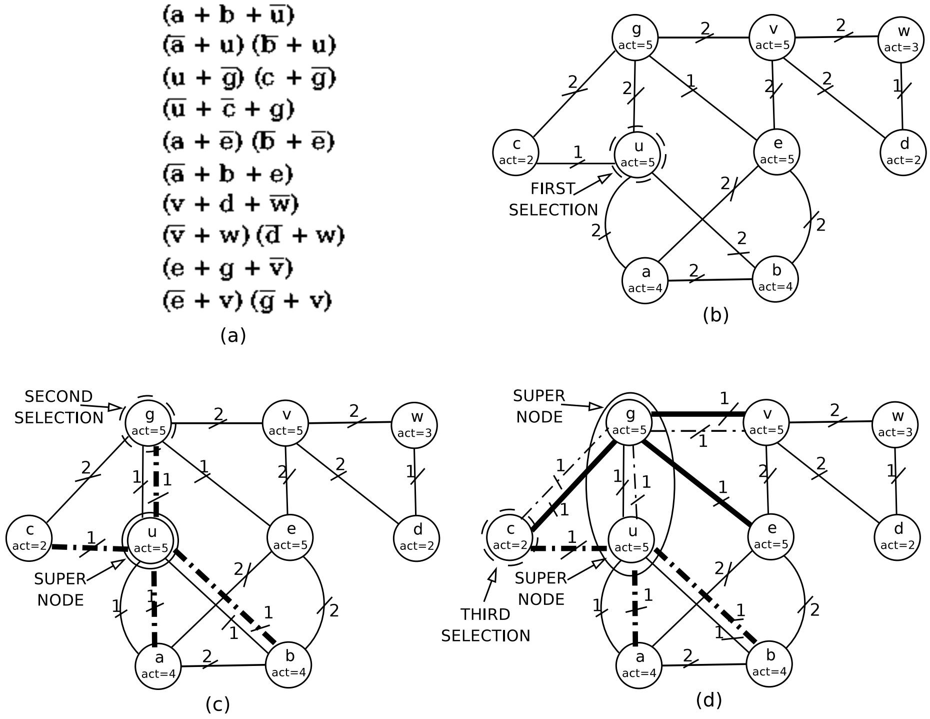
To overcome the above limitations , this paper presents a new approach to derive an initial static ordering for SAT search by rigorously analyzing constraint-variable dependencies. Moreover, we analyze the effect of decision-assignments on the variable order and exploit this effect to further improve the order. 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).

2 ACCORD: Activity-Correlation Based Ordering

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 [1]. 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 2.1. *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**.

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



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

Fig. 1. An example CNF and its Weighted Graph - Edge weights denote the degree of correlation between the variables (vertices). Variable activity depicted within the node

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(a). Its corresponding weighted graph is depicted in Fig. 1(b).

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 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. 1(b) as "First Selection." The variable u becomes the supernode. Now we need to identify the set of variables connected to this supernode u. Note that, when the solver branches on this variable u, it will assign $u = 1$. This is because the activity of its positive literal is greater than that of negative. Hence, all the clauses corresponding to the literal u will be satisfied due to this assignment. In order to exploit this behaviour, our algorithm determines connectivity only from the clauses in which literal $\bar{u}$ appears. For this purpose, the incident edges on the node u are split into two edges corresponding to the literals and their correlation as shown in Fig. 1(c). The dotted edge corresponds to the negative literal ($\bar{u}$). All the nodes that share at least one edge with the supernode ($\bar{u}$) are identified. The nodes $\{g, c, a, b\}$ are connected to the literal $\bar{u}$, as shown in Fig. 1(c). 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, the variables $\{g, a, b, c\}$ have the same degree of correlation with node u, which is equal to 1. 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 $\{a, b, c\}$, g is selected as the next branching variable. This is shown in Fig. 1(c) 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* $= \{u, g\}$.

Algorithm 1. Pseudo code for ACCORD

```
1: INPUT = CNF Clauses and Variable-Clause Database ;
2: /* For each variable $x_i$, the Variable-Clause database contains a list of clauses in
   which $x_i$ appears */
3: OUTPUT = Variable order for SAT search ;
4: activity_list = Array containing activity of each variable ;
5: var_order_list = Initialize variable order according to activity ;
6: connectivity_list = Initialize to zero for all variables ;
7: for (i=0; i != number of variables; i++) do
8:    /* Implicitly, supernode = {var_order_list[0], ..., var_order_list[i]} */
9:    next_var = var_order_list[i] ;
10:   correlation_list = find variables connected to least active literal of next_var using
      Variable-Clause database ;
11:   for all var $\in$ correlation_list do
12:      connectivity_list[var]++ ; /* Compute correlation */
13:   end for
14:   adjust_variable_order(var $\in$ correlation_list) ;
15:   /* Linear sort is used to update the variable order corresponding to both con-
      nectivity_list as well as activity_list */
16:   /* Here, {var_order_list[0], ..., var_order_list[i]} is the current variable order*/
17: end for
18: return(var_order_list) ;
```

As the next step, those variables are identified that share an edge with the supernode in the set of unresolved clauses. The activity of the literal $\bar{g}$ is greater than the literal g. Hence, only those clauses in which g appears are considered, as they are unsatisfied. These variables are $\{v, e, c, a, b\}$ as shown in Fig.1(d) with highlighted incident edge. Consider the nodes $\{v, e, a, b\}$. They share only one edge with the supernode with weights (correlation) = 1. 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 $\{g, u\} = 1{+}1 = 2$. Since this sum is the highest for c, it is selected as the next branching variable. This is shown in Fig. 1(d) 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 [13]. We have named it the **ACCORD** (ACtivity COrrelation ORDering) algorithm. The Pseudo code for the ACCORD algorithm is presented in Algorithm 1. The Variable-Clause database, mentioned in lines 1-2, is implemented using the array of arrays data-structure available within contemporary SAT solvers [2] [14]. 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. 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.

3 Experimental Results and Analysis

The ACCORD algorithm has been programmed within the zCHAFF [2] solver using its native data-structures. The algorithm analyzes 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 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 [15]; 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.

Table 1 presents some results that compares the quality of the variable order derived by ACCORD with those of zCHAFF (latest version developed in 2004) and hypergraph partitioning (HGPart) based order [12]. 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 ACCORD results in a orders of magnitude speedup in SAT solving. Particularly for more difficult problems, our approach significantly outperforms the other two.

Table 1. Run-time Comparison of ACCORD with zCHAFF and HGPART

		zCHAFF	HGPart + zCHAFF			ACCORD + zCHAFF		
Bench-mark	Vars/ Clauses	Solve (sec)	Var. Time(s)	Solve Time(s)	Total Time(s)	Var. Time(s)	Solve Time(s)	Total Time(s)
3bitadd_31	8432/21K	68.88	3.023	0.55	**3.573**	0.09	4.1	4.19
3bitadd_32	8704/32K	7.43	3.417	0.67	4.087	0.1	0.02	**0.12**
clus-2020-1	1200/4800	>1000	1.421	>1000	–	0.01	666.32	**666.33**
clus-2020-2	1200/4800	688.19	1.81	9.12	**10.93**	0.04	37.17	37.21
clus-1010-3	1200/4919	>1000	1.701	704.66	706.361	0.01	700.85	**700.86**
color-10-3	300/6475	129.57	0.375	19.01	**19.385**	0	25.05	25.05
conn-939	576/6864	135.54	1.196	28.96	30.156	0	0.42	**0.42**
conn-945	596/7157	581.56	1.145	>1000	–	0.01	98.02	**98.03**
mm-1x10	1120/7220	654.16	1.057	>1000	–	0.01	237.48	**237.49**
qwh-35	1597/11K	39.73	1.869	84.59	86.459	0.03	5.58	**5.61**
unif-r4-2	500/2000	>1000	0.447	>1000	–	0	198.47	**198.47**
unif-r4-5	500/2000	884.04	0.491	91.88	**92.371**	0	129.79	129.79
unif-r4-7	500/2000	>1000	0.451	582.53	582.981	0	151.25	**151.25**
unif-r4-9	500/2000	>1000	0.493	711.65	712.143	0	141.95	**141.95**
ferry10	2958/21K	3.54	3.549	0.73	4.279	0.04	0.07	**0.11**
ferry12	4222/32K	238.5	8.189	134.78	142.969	0.09	66.51	**66.6**
icos-stretch	45/352	102.99	0.199	93.84	94.039	0	78.56	**78.56**
marg33-ch	41/272	247.73	0.071	142.53	142.601	0	20.01	**20.01**
marg33add8	41/224	3.44	0.074	1.36	1.434	0	1.29	**1.29**
marg35	61/280	>1000	0.085	>1000	–	0	289.17	**289.17**
mm-2x2-50	60/32000	39.8	3.815	31.37	35.185	0.05	24.9	**24.95**
mm-2x3-66	1698/49K	17.32	4.386	16.48	20.866	0.08	13.55	**13.63**
urqh1c3x3	41/204	284.48	0.078	8.92	**8.998**	0	57.96	57.96
urqh2x4	42/336	302.79	0.077	71.07	71.147	0	26.53	**26.53**
urqh2x5	53/432	>1000	0.117	>1000	–	0	612.96	**612.96**
urqh1c3x4	58/476	>1000	0.143	196.78	**196.923**	0	320.62	320.62
rotmul	5980/35K	174.7	5.888	153.26	**159.148**	0.28	159.53	**159.81**
9dlx-iq1	25K/261K	504.47	68.83	443.79	512.62	15.69	381.09	**396.78**

Table 2 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 ACCORD is used. Note that, using the 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 proposed by [3]. As a result for these benchmarks, the order generated by ACCORD gets significantly modified due to the conflict clause resolution [3] and hence, the impact of ACCORD is minimal.

Table 2. Run-time Comparison of ACCORD with zCHAFF'03 and zCHAFF'04

		zCHAFF 2003	ACCORD+ zCHAFF'03			zCHAFF 2004	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	5K/80K	129	0.56	56.1	56.66	8.02	11.96	12.52
5pipe	9K/195K	196.53	2.36	54.46	56.82	18.34	16.59	18.95
4pipe_k	5K/79K	208.81	0.6	40.85	41.45	8.34	14.53	15.13
5pipe_k	9K/189K	871.79	2.25	388.32	390.57	40.75	31.29	33.54
4pipe_q0	5K/69K	66.71	0.48	73.86	74.34	6.37	8.59	9.07
5pipe_q0	10K/154K	649.9	1.53	214.89	216.42	25.21	26.46	27.99
6pipe	16K/394K	>2000	6.53	827.06	833.59	132.82	144.72	151.25
6pipe_k	15K/408K	105.52	8.76	141.45	150.21	71.49	57.64	66.4
6pipe_q0	17K/315K	104.48	4.88	82.44	87.32	43.37	46.65	51.53
7pipe_q0	26K/536K	>2000	12.45	1523.16	1535.61	284.97	265.3	277.75

4 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 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. J. P. M. Silva, "The Impact of Branching Heuristics in Propositional Satisfiability Algorithms", *in Portuguese Conf. on Artificial Intelligence*, 1999.
2. 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.
3. E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust Sat-Solver", *in DATE, pp 142-149*, 2002.
4. J. Marques-Silva and K. A. Sakallah, "GRASP - A New Search Algorithm for Satisfiability", *in ICCAD'96*, pp. 220–227, Nov. 1996.
5. R. Dechter and J. Pearl, "Network-based Heuristics for Constraint-Satisfaction Problems", *Artificial Intelligence*, vol. 34, pp. 1–38, 1987.
6. 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.

7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. V. Durairaj and P. Kalla, “Guiding CNF-SAT Search via Efficient Constraint Partitioning”, *in ICCAD*, pp. 498 – 501, 2004.
13. R. C. Prim, “Shortest connection networks and some generalizations”, *The Bell System Technical Journal*, vol. 36, pp. 1389–1401, 1957.
14. N. Eén and N. Sörensson, “An Extensible SAT Solver”, *in 6th International Conference, SAT*, 2003.
15. M. Velev, “Sat benchmarks for high-level microprocessor validation”, http://www.cs.cmu.edu/ mvelev.