Dynamic Analysis of Constraint-Variable Dependencies to Guide SAT Diagnosis

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

Submitted to HLDVT 2004

Topic Category: Boolean Satisfiability Tools, Design Validation and Formal Verification Methods

Designated Contact Author: Vijay Durairaj
Department of Electrical and Computer Engineering
50 S. Central Campus Drive, MEB 3280
University of Utah
Salt Lake City, UT-84112
Ph: (801)-864-9306
Fax: (801)-581-5281
Email: durairaj@ece.utah.edu

Dynamic Analysis of Constraint-Variable Dependencies to Guide SAT Diagnosis

Abstract: An important aspect of the Boolean Satisfiability problem is to derive an ordering of variables such that branching on that order results in a faster, more efficient search. Contemporary techniques employ either variable-activity or clause-connectivity based heuristics, but not both, to guide the search. This paper advocates for simultaneous analysis of variable-activity and clause-connectivity to derive an order for SAT search. Preliminary results demonstrate that the variable order derived by our approach can significantly expedite the search.

As the search proceeds, clause database is updated due to added conflict clauses. Therefore, the variable activity and connectivity information changes dynamically. Our technique analyzes this information and re-computes the variable order whenever the search is restarted. Preliminary experiments show that such a dynamic analysis of constraint-variable relationships significantly improves the performance of the SAT solvers. Our technique is very fast and this analysis time is a negligible (in milliseconds) even for instances that contain a large number of variables and constraints. This paper presents preliminary experiments, analyzes the results and comments upon future research directions.

I. Introduction

The Boolean Satisfiability problem (henceforth called SAT) is one of the pivotal problems in the Electronic Design Automation (EDA) arena. SAT is the problem of finding a solution (if one exists) to the equation $f = \mathbf{1}$, where f is a Boolean formula to be satisfied. Classical approaches to CNF-SAT are based on variations of the well known Davis-Putnam (DP) [1] and Davis-Logemann-Loveland (DLL) [2] procedures. Recent approaches [3] [4] [5] etc., employ sophisticated methods such as constraint propagation and simplification, conflict analysis, learning and non-chronological backtracks to efficiently analyze and prune the search space. This paper proposes a new decision heuristic that analyzes constraint-variable dependencies to guide SAT diagnosis. Branching on the variable order derived by our approach significantly improves the performance of SAT solvers.

A. Related Work:

Over the years, a lot of effort has been invested in **deriving an ordering of variables** such that branching on that order results in a faster, more efficient search for solutions. **Variable activity** and **clause connectivity** statistics are exploited as qualitative and quantitative metrics to guide the search. Activity of a variable (or literal) is defined as the number of its occurrence among all the clauses of a given SAT problem. SAT solvers compute the activity of variables and perform the search by case-splitting on variables of high activity. Contemporary tools such as zCHAFF, BerkMin, etc. dynamically update the activity of the variables as and when conflict clauses are added to the original constraints. For a comprehensive review of the effect of activity-based branching strategies on SAT solver performance, reviewers are referred to [6].

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 [7] [8] for analyzing connectivity of constraints in constraint satisfaction programs (CSP). Such techniques have also found application in Boolean satisfiability problems. It has been shown [9] [10] that identifying **minimum tree-width** for the decomposed tree structures results in partitioning the overall problem into a chain of connected constraints. MINCE [11] employs CAPO placer's mechanism [12] to find a variable order such that the clauses are resolved according to their chain of connectivity. Our recent work [13] employs hypergraph partitioning methods to derive a tree decomposition. Various other approaches operate on such partitioned tree structures by deriving an order in which the partitioned set of constraints are resolved [14] [15] [10].

B. Limitations:

Contemporary methods that guide SAT diagnosis have the following limitations:

- Conventional SAT solvers [4] [5] [3] employ variable activity-based branching heuristics (DLIS, VSIDS, etc.) to resolve the constraints. On the other hand, partitioning/tree-decomposition based approaches [16] [10] [15] [11] employ clause-connectivity based branching heuristics. None of these techniques utilize both activity and connectivity information *simultaneously* for SAT diagnosis.
- The high computational complexity of the tree decomposition/partitioning based approaches results in large compute times to search for the variable order. As a result, these techniques are somewhat impractical for solving large and hard CNF-SAT problems [10] as often encountered in design and validation problems in VLSI-CAD.
- Other approaches rely on the sophistication of hypergraph partitioning [17] [13] and place-and-route mechanisms [11] to derive a variable order for SAT search.
- As the search proceeds, conflicts are encountered and conflict-induced clauses are added to the constraint database. Thus, the activity-connectivity information changes dynamically. While activity based heuristics update this information (VSIDS, DLIS, etc.), connectivity based variable orders are de-

rived only statically.

- Contemporary partitioning based methods are not suitable to be employed dynamically, mostly because of their time complexity[1]. The problem is exacerbated due to a significant increase in the clause database due to added conflict clauses.

C. Contributions of this research:

Efficient CNF-SAT decision heuristics should analyze both variable activity and clause connectivity simultaneously so as to exploit constraint-variable relationships for faster constraint resolution. Moreover, the updated (due to conflict clauses) variable activity and clause connectivity information should be utilized dynamically during the search process. Furthermore, the time to analyze constraint-variable relationships should be a small fraction of the overall solving time. This paper proposes an efficient technique to guide SAT diagnosis that attempts to fulfill the above criteria/requirements.

Before proceeding into the search, our approach analyzes high activity variables and identifies the clauses in which they appear. These clauses contain other variables that "connect/link" to these high activity variables. This connectivity information is extracted, in *decreasing order of variable activity*, from the entire clause-variable database. Iterative application of the above procedure produces an order for SAT search. This analysis is repeated and a new order is derived every time the search is *restarted*. Our variable order generation procedure is generic and can be implemented within any SAT solver. Moreover, the time to compute the order is negligible, even for large instances. Furthermore, our technique improves the performance of SAT engines by orders of magnitude.

II. Analyzing Constraint-Variable Dependencies

It is our desire to derive a variable order for SAT search by analyzing clause-variable relationships. To achieve this, we propose a constraint decomposition scheme by simultaneously analyzing variable-activity as well as clause connectivity. We begin the search for such an order by first selecting the highest active variable and store it in a list (*var_ord_list*). The SAT tool should branch on this variable first. Now, we need to identify the set of variables related (connected/dependent) to this highest active variable. This information can be obtained by analyzing all the clauses in which the highest active variable appears. Such clauses are identified and marked. These clauses act as a chain connecting the highest active and its related variables. These related variables are termed as Level-1 connectivity variables. Subsequently, level-1 connectivity variables are ordered according to their activity in the remaining problem (unmarked clauses) and appended to the *var_ord_list*. The reason for ordering level-1 connectivity variables according to their activity in the remaining problem (as opposed to their overall activity) is because they might be implicated due to any decision on the highest active variable.

[1]Minimum tree width decomposition algorithms[7] [9] are known to be time exponential in the tree width.

Variables related to the level-1 connectivity variables are to be identified next. For this purpose, the clauses corresponding to level-1 connectivity variables are analyzed and the level-2 connectivity variables are extracted. These level-2 connectivity variables are also ordered according to their activity in the remaining problem and appended to the *var_ord_list*. This procedure is applied iteratively until all the variables are ordered. This order is used by the SAT engine to resolve the constraints. This procedure can be visualized as shown in Fig. 1.

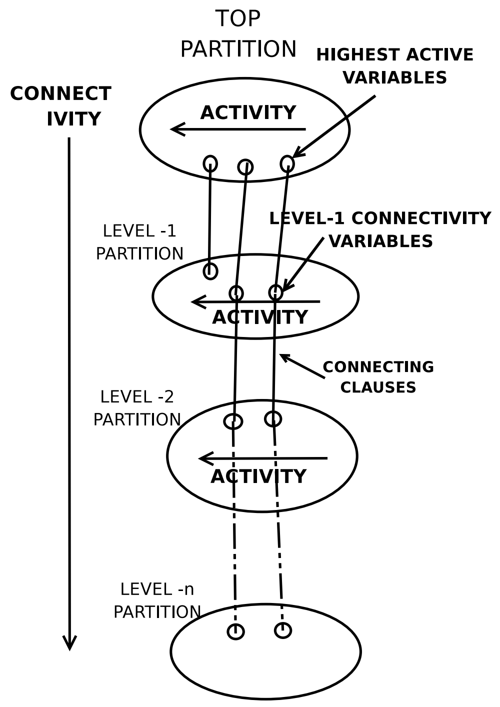

Fig. 1. Constraint Decomposition analyzing constraint-variable relationship

In general, there might be more than one variables that have the same highest activity measure. In such cases, we need to decide whether to select just one of them, all of them, or a subset of the highest active variables as the top-level partition. It is difficult to answer this question analytically; however, this decision affects the variable order inasmuch as it affects the granularity of the decomposition. To elaborate this issue further, let us analyze the application of our algorithm for the circuit shown in the Fig. 2.

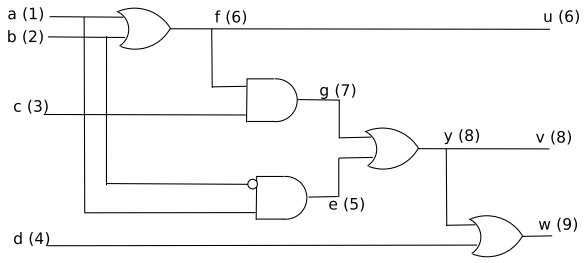

Fig. 2. An Example Circuit : The integer values next to the variable names correspond to the mapped literal in the CNF file.

Suppose that it is required to solve the satisfiability problem for the above circuit. This circuit-SAT problem can be formulated as CNF-SAT by generating clauses for the gates. It can be observed from the circuit that the activity of variables {7, 5, 6, 8} is the highest. If we select only the variable 7 as the top-level variable, then the resulting decomposition would look like as shown in Fig. 3(a). The generated variable order would be {7, 6, 5, 8, 3, 4, 9, 1, 2}. Note that variables {4, 9} have the lowest activity in the overall problem, however, they are ordered before {1, 2}. This is because our procedure orders the variables according to their activity in the unmarked clauses. If we select two of the highest active variables, say 7 and 5, then the decomposition changes as shown in Fig. 3(b). The resulting variable order would be {7, 5, 6, 8, 1, 2, 3, 4, 9}; this is a different order than the previous one.

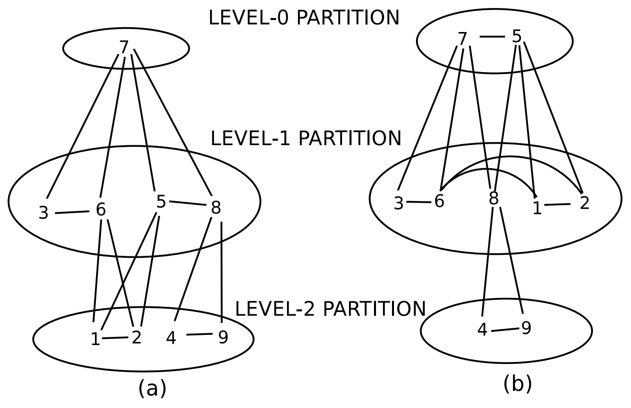

Fig. 3. Constraint-Variable relationship and SAT search order

To come back to our question of how many top variables to select, we ran experiments with different threshold values for the number of top-level variables. In particular, we experimented with: 1) just one top variable; 2) 5% of total variables; 3) 7%; 4) 10%; 5) 12%; and 6) all the variables with the highest activity. Results were inconclusive for all cases except when 5% and 7% of the total variables are selected for the top-level partition. Only in these cases, we found that the derived variable provided consistent improvement in performance of the SAT engine. For the other cases, the results were mixed. Now we present the results for these two cases (5% and 7% top-level threshold) and analyze them further.

III. Preliminary Experimental Results and Analysis

The proposed algorithm has been implemented within the diagnosis engine of the MiniSAT solver [18]. The choice for MiniSAT was dictated by its open source code, efficient resolution procedures (it outperforms zCHAFF and GRASP on many large instances) and also because it implements intelligent mechanisms to invoke *search re-starts*. Our implementation modifies the variable ordering scheme of MiniSAT. Using our modifications, we ran experiments on satisfiable and unsatisfiable instances selected from a range of applications. There are a large number of CNF-SAT instances that are easily solved by MiniSAT - in less than a minute. We wanted to analyze the robustness of our approach by experimenting with problems of large size and difficult nature. Therefore, the benchmarks selected for experiments are those which take a long time to solve - where there exists enough scope for improvements. These experiments are depicted in Table I.

In the table, the run times of original MiniSAT solver are compared with those obtained by the modified version of our decision heuristics. Threshold values of 5% and 7% correspond to the order derived by selecting 5 or 7 percent of the total variables in the top-level partition. At every search restart, we re-analyze the updated clause-variable dependencies to recompute the variable order. The CPU times reported for our procedure include both the time to compute the variable order, as well as the time to resolve the constraints. It can be observed that our modifications significantly outperform the original MiniSAT implementation.

TABLE I
Run-time Comparison of our proposed approach with MiniSAT

			Modified MiniSAT	
Bench-mark	Vars/ Clauses	MiniSAT Time(sec)	5% Time(s)	7% Time(s)
Urq3_5	43 / 334	184.13	**130.46**	154.2
hanoi5	1931 / 14468	40.65	**18.69**	49.77
color_10_3	300 / 6475	81.63	8.72	**4.11**
clus_set1_1	1200 /4800	242.73	**11.37**	170.64
c5315_opt	4992 / 14151	56.65	**19.52**	32.96
4pipe	5237 / 80213	139.57	**42.76**	155.11
4pipe_k	5095 / 79489	186.91	**92.7**	218.81
4pipe_q0_k	5380 / 69072	**48.04**	49.56	53.21
engine_4	6944 / 66654	62.59	53.32	**48.3**
5pipe	9471 / 195452	68.29	**40.8**	63.11
5pipe_k	9330 / 189109	1362.06	956.16	**397.15**
5pipe_q0_k	10026 / 154409	653.82	618.37	**123.27**

Let us now comment on the selection of threshold values. It can be observed from the table that for instances that have fewer than 7000 variables, a threshold value of 5% results in better performance. Whereas for instances with larger than 7000 variables, a larger threshold value of 7% provides better results. This result is intuitive and not surprising. For smaller problems, a more fine-grained decomposition provides a better variable order. On the other hand, for large problems, the performance gain might be offset due to larger computational overheads when the same fine-granularity decomposition is employed. Based on these observations, we conjecture that perhaps a low threshold value is better suited for smaller size problems and a higher one for larger problems. This suggests that our variable ordering scheme should be adaptive to the problem size. Based on the number of variables, we should automatically decide on what threshold to choose.

Amir's approach [16] tries to resolve a similar problem by analyzing clause-to-variable ratio. Unfortunately, for our experiments we are not able to derive any conclusions regarding the threshold value based on the clause-to-variable ratio. However, this work is currently in progress and this issue requires

TABLE II
VARIABLE ORDER RE-COMPUTATION AT SEARCH RESTARTS

		Static				Dynamic			
Bench-mark	% Thres-hold	Rest-arts	Conf-licts	Deci-sions	Time (sec)	Rest-arts	Conf-licts	Deci-sions	Time (sec)
Urq3_5	5	27	7.64 M	11.1 M	143.88	26	6.8 M	9.9 M	130.46
hanoi5	5	16	107K	244 K	18.81	16	98 K	219 K	18.69
color_10_3	5	14	40 K	57 K	4.1	15	70 K	96 K	8.72
clus_set1_1	5	23	1.88 M	2.96 M	561.59	15	70 K	133 K	11.37
c5315_opt	5	14	44 K	107 K	12.74	14	57 K	185 K	19.52
4pipe	5	15	74 K	257 K	76.49	15	65 K	257 K	42.76
4pipe_k	5	15	66 K	482 K	58.57	15	64 K	334 K	92.7
4pipe_q0_k	5	16	104 K	346 K	61.2	16	103 K	400 K	49.56
engine_4	7	14	41 K	95 K	54.38	14	42 K	121 K	48.3
5pipe	7	15	69 K	514 K	89.93	14	57 K	777 K	63.11
5pipe_k	7	–	–	–	>1000	18	204 K	885 K	397.15
5pipe_q0_k	7	19	364 K	1.29 M	405.77	17	179 K	790 K	123.27

further research.

A. Search Restarts: Re-computation of the Variable Order

We now highlight the importance of *dynamically* updating/re-computing the variable order. MiniSAT automatically invokes search restarts after a certain number of conflicts are encountered. As and when conflicts are encountered, conflict-induced clauses are added to the database. This, in turn, changes both variable activity and clause connectivity. Therefore, this updated information should be re-analyzed as and when the search is re-started. Table II depicts the performance improvements when the clause-variable relationship is analyzed and exploited dynamically. The columns under the "Static" heading contains run-time statistics when the order is computed only once at the beginning of search and not re-computed during restarts. On the other hand the columns under the "Dynamic" heading show run-time statistics when the variable order is recomputed at search restarts. Note that this dynamic clause-variable analysis almost always improves the performance of the solver.

IV. CONCLUSIONS AND RESEARCH STATUS

This paper has advocated for the need to analyze constraint-variable relationship to derive a variable order to guide SAT diagnosis. For this purpose, we have proposed a algorithm that decomposes the constraints by analyzing variable activity together with clause connectivity to derive a variable order. Our approach is very fast, scalable and improves the performance of the SAT engine. Preliminary results have been promising and encourage further research.

Status of our work: Clearly, a missing piece of the puzzle is to identify a definitive threshold value, along with an adaptive strategy, to derive a variable order according to the problem size and/or constraindness. This issue is under investigation and we would value reviewers recommendations. We are also implementing a dynamic variable order update strategy to be employed when conflict clauses are added to the database. Note that this is as opposed to the variable order re-computation at search re-starts.

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] J. P. M. Silva, "The Impact of Branching Heuristics in Propositional Satisfiability Algorithms", *in Portuguese Conf. on Artificial Intelligence*, 1999.
[7] R. Dechter and J. Pearl, "Network-based Heuristics for Constraint-Satisfaction Problems", *Artificial Intelligence*, vol. 34, pp. 1–38, 1987.
[8] R. Dechter, *Constraint Processing*, chapter 9, Morgan Kaufmann Publishers, 2003.
[9] E. Amir, "Efficient Approximation for Triangulation of Minimum Treewidth", *in 17th Conference on Uncertainty in Artificial Intelligence (UAI '01)*, 2001.
[10] 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.
[11] 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.
[12] A. Caldwell, A. Kahng, and I. Markov, "Improved Algorithms for Hypergraph Bipartitioning", *in In Proc. Asia-Pacific DAC*, 2000.
[13] V. Durairaj and P. Kalla, "Exploiting Hypergraph Partitioning for Efficient Boolean Satisfiability", *in submitted, in review, HLDVT*, 2004.
[14] E. Amir and S. McIlraith, "Partition-Based Logical Reasoning", *in 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'2000)*, 2000.
[15] 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.
[16] 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.
[17] D. Wang, E. Clark, Y. Zhu, and J. Kukula, "Using Cutwidth to Improve Symbolic Simulation and Boolean Satisfiability", *in In Proc. High-Level Design Validation and Test Workshop*, Nov. 2001.
[18] N. Eén and N. Sörensson, "An Extensible SAT Solver", *in 6th International Conference, SAT*, 2003.