Guiding CNF-SAT Search via Efficient Constraint Partitioning

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: Contemporary techniques to identify a good variable order for SAT rely on identifying minimum tree-width decompositions. However, the problem finding a minimal width tree decomposition for an arbitrary graph is NP complete. The available tools and methods are impratical, as they cannot handle large and hard-to-solve CNF-SAT instances. This paper proposes a novel hypergraph partitioning based constraint decomposition technique as an alternative to contemporary methods. We model the CNF-SAT problem on a hypergraph and apply min-cut based bi-partitioning. Clause-variable statistics across the partitions are analyzed to further decompose the problem, iteratively. The resulting tree-like decomposition provides a variable order for guiding CNF-SAT search. Experiments carried out over a large and varied set of benchmarks demonstrate that our partitioning procedure is very fast and scalable. The variable order derived through the partitioning results in significant increase in performance (often orders of magnitude) of the SAT engine.

I. Introduction

The Boolean Satisfiability problem (henceforth called SAT) is one of the pivotal problems in the Electronic Design Automation (EDA) arena and Artificial Intelligence (AI). Applications in EDA include functional verification, bounded and unbounded model checking, combinational equivalence checking, logic synthesis, among others. SAT belongs to the class of NP-complete problems for which the algorithmic solutions exhibit exponential worst-case complexity [1].

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. The formula (f) can be represented in Conjunctive Normal Form (CNF), or with Binary Decision Diagrams (BDDs)[2]. Classical approaches to CNF-SAT are based on variations of the well known Davis-Putnam (DP) [3] and Davis-Logemann-Loveland (DLL) [4] procedures. Typical versions of the above [5] [6] [7] [8] incorporate a chronological backtrack-based search that, at each node in the search tree, selects an assignment and prunes subsequent search by iterative application of the unit clause and pure literal rules [9]. Recent approaches [10] [11] [12] [13] etc., employ sophisticated methods such as constraint propagation and simplification [9], conflict analysis [10], learning [14] and non-chronological backtracks [10] [12] [13] to efficiently analyze and prune the search space. Binary Decision Diagrams (BDDs) [2] [15] have also been used to solve the SAT problem. However, for large designs, BDDs suffer from the well-known size explosion problem. As a result, use of BDD-based SAT tools [16] [17] has been limited to problems of relatively small size.

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 [18] [19], local search strategies [20], complexity of SAT *viz-a-viz* ATPG [21], relationship of BDD variable orderings and CNF search procedures [22], the amount of search space analyzed [23], UNSAT core extraction [24] [25], among others [26] [17] [16].

Over the years, constraint partitioning and tree-decomposition based approaches have been investigated in the context of constraint satisfaction problems [27] [28] [29]. Recently, such approaches have also found application in VLSI-CAD; particularly with respect to those problems that can be modeled as DPLL-based CNF-SAT search [29] [30] [31] [32]. An important aspect of CNF-SAT search procedures is to **derive an ordering of variables** such that branching on that order results in a faster, more efficient search for solutions. Above approaches analyze and exploit the variable-constraint relationship to derive such an order for efficient SAT search. However, the computational complexity of the proposed algorithms 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 [31].

This paper proposes partitioning based SAT search procedures that attempt to overcome the practical limitations of the above approaches. Hypergraph based constraint partitioning techniques are employed to derive the tree decomposition. Variable activity and clause connectivity statistics are analyzed to make the partitioning efficient. The proposed technique is scalable and can operate on large set of variables and constraints efficiently. We show that the variable order derived by our technique improves the SAT solver performance by one or more orders of magnitude.

II. Previous work in partitioning and tree decomposition

Boolean functions arising in many applications represent some spacial, casual or logical dependencies (or connections) among variables. Therefore, processing these "connected" functions together, and "disjoint" functions separately, seems intuitively justified. This suggests that we can decompose the given SAT problem into relatively independent groups (partitions). Variables that may appear in more than one partition would model the "connections" or dependencies among the clauses. This motivates the use of constraint partitioning and decomposition techniques to facilitate efficient search procedures for CNF-SAT problems.

Analysis of clause-variable dependencies provides useful information that can be exploited for partitioning based SAT approaches. **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. For a comprehensive review of the effect of activity-based branching strategies on SAT solver performance, reviewers are referred to [33].

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 [27] [28] for analyzing connectivity of constraints in constraint satisfaction programs (CSP). It has been shown [34] [31] that identifying **minimum tree-width** for the decomposed tree structures results in partitioning the overall problem into a chain of connected constraints. MINCE [22] employs CAPO placer's mechanism [35] to find a variable order such that the clauses are resolved according to their chain of connectivity. Various approaches operate on such partitioned tree structures by deriving an order in which the partitioned set of constraints are resolved. For example, this order can be computed according to the **degree of constrainedness** of sub-problems [29] [30]. Other related works are: (i) guiding SAT diagnosis based on tree decompositions [31]; (ii) partition based decision heuristics [32]; (iii) decomposable negation normal form [36] [37].

The order in which variables (and correspondingly, constraints) are resolved significantly impacts the performance of SAT search procedures. Most conventional SAT solvers [12] [13] [10] employ activity-based branching heuristics (DLIS, VSIDS, etc.) to resolve the constraints. On the other hand, the tree decomposition based approaches derive the variable order using either clause connectivity [30] [22] [31] or variable activity [32], *but usually not both of them simultaneously*. In the following section, we show, by means of an example, that analysis of variable activity or clause connectivity alone may not result in a sufficiently robust and efficient SAT search. Subsequently, we motivate the need to analyze variable activity along with clause connectivity, to derive a tree decomposition. This tree decomposition, in turn, suggests a good variable order for efficient SAT search.

III. On The Need to Analyze Variable Activity and Clause Connectivity Simultaneously

Consider the circuit shown in Fig. 1. It is required to generate a functional/simulation vector at the inputs that would excite the values $u = 0, v = 1, w = 1$ at the primary outputs. This problem can be formulated as a CNF-SAT problem by generating clauses for the gates, along with clauses for the constraints corresponding to the required output values. It can be observed that no such vector exists that can satisfy the required output constraints. Hence the given problem is unsatisfiable (UNSAT).

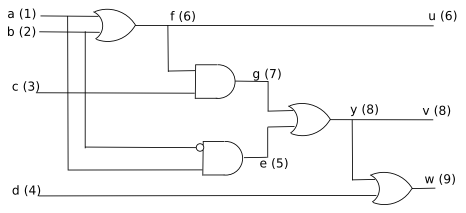

Fig. 1. An Example Circuit : The integer values next to the variable names correspond to the mapped literal in the CNF fi le.

Let us analyze the application of contemporary partitioning-based procedures to derive a variable order for the example shown in Fig. 1. Consider, first, the join-tree-clustering (JTC) algorithm by Dechter *et al.* [27] [28]. The algorithm converts the given problem to a *hypergraph*, in which the variables form the vertices and clauses form the hyperedges. This graph is then converted to an *induced chordal graph*[1], from which, the maximal cliques are identified. A *join-tree* structure is created using these cliques. This join-tree is the *tree-decomposition* of the given problem, which is shown in Fig. 2. As shown in the figure, each node corresponds to a set of clauses (partitioning). The support variables of these clauses are depicted within the node. For example, the top node contains clauses with variables $\{1, 2, 6, 5\}$. The variables labeled on the edges denote the connectivity - these variables are common to adjacent nodes. This decomposed problem is then solved using either the cluster-tree-elimination (CTE) or adaptive-tree-consistency (ATC) algorithm [28].

The above technique is geared towards reducing the tree-width of the derived graph. The reduction in the tree-width subsequently corresponds to modeling the constraint partitions according to clause connectivity. Based on the decompositions shown in Fig. 2, a variable order can be derived for SAT search by traversing the tree top-down or bottom-up. For example, traversing the tree top-down produces the following order: $\{1, 2, 6, 5, 7, 3, 8, 4, 9\}$. The MINCE [22] technique generates an order that is the opposite of the above - $\{4, 9, 8, 3, 7, 5, 6, 1, 2\}$. The recent work of [31] also uses

[1] A graph is chordal if every cycle of length atleast 4 has a chord, that is, an edge connecting two non-adjacent vertices.

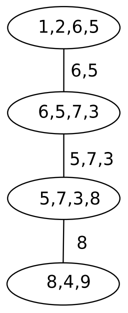

Fig. 2. Rina Dechter's Method

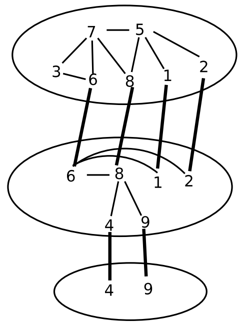

Fig. 3. Proposed Decomposition

MINCE's variable order to produce its tree-based decomposition.

It can be observed from the figure that clauses corresponding to variables 4 and 9 do not contribute to the unsatisfiability of the problem. Variable order generated by MINCE would branch on these variables first, which is not necessary. On the other hand, the order generated by traversing the tree top-down has the variables 4 and 9 appearing at the end, and hence may lead to an efficient search. This begs the question that given a partitioning, which order to choose such that search can be performed efficiently? It is not easy to answer this question by analyzing the tree decomposition because the partitioning was derived solely on clause connectivity.

This limitation can be overcome by simultaneously analyzing both variable activity and clause connectivity to generate the tree decomposition. For example, a decomposition analyzing both variable activity and connectivity is shown in Fig. 3, with the variable order $\{7, 5, 6, 8, 1, 2, 3, 9, 4\}$ generated by traversing the tree top-down (always). The procedure to generate such a variable order is elaborated in subsequent sections of the paper.

The authors wish to point out that the above example is used only for didactic purposes to highlight the fact that constraint partitioning/decomposition approaches based solely on variable activity or clause connectivity may not be sufficiently robust. Through the above example, we just want to motivate the need for simultaneous activity/connectivity analysis.

A. *Time complexity limitations of previous work*

It is well-known [31] that the computational complexity of contemporary tree-decomposition methods is too expensive to be applicable for large CAD problems. The tree-decomposition algorithms by Dechter et. al. [27] [28] are shown to be time exponential in the tree-width. The time complexity of JTC algorithm is $O(r \cdot k^{w*(d)+1})$, where *r* is the number of constraints, *k* is the maximum domain size and *w*(d)* is the induced width of the ordered graph. Amir *et al.* [38] [30] have developed much more efficient algorithms for approximating tree-width with bounded error [34], but even these appear to be too costly for industrial problems.

Partitioning based approaches that are used to derive good variable ordering for SAT search, should be fast and robust enough to handle a large set of variables and constraints. This is particularly important for design validation problems in VLSI-CAD that often have a significantly large number of variables and clauses [39]. 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, *we have observed that MINCE [40] and Amir's tools [38] require unacceptably long time just to derive the variable order*. Similarly, the technique of [31] is based upon MINCE and Amir's computational engines - as such, it also suffers from large compute times.

B. *Research Contributions:*

The primary goal of this research is to identify a variable ordering that would result in efficient and robust CNF-SAT search. Contemporary techniques to identify a good variable order for SAT rely on identifying minimum tree-width decompositions. Finding a minimal width tree decomposition for an arbitrary graph is NP complete [41]. Therefore, a very efficient algorithm that would operate on large CNF-SAT instances is unlikely to exist. Indeed, contemporary heuristic-based tree decomposition methods have also been shown to be impractical [31]. Clearly, there is a need to find an alternative to minimum tree-width decomposition schemes. This paper suggests a solution based on hypergraph partitioning and refinement procedures to derive a tree decomposition.

The CNF-SAT problem is modeled on a hypergraph and min-cut based bi-partitioning techniques are employed. Clause-variable statistics across the partitions are analyzed and a sub-problem corresponding to high activity variables is extracted. Subsequently, a novel, iterative partitioning procedure is employed that analyzes the clause connectivity, in decreasing order of variable activity, to decompose the constraints in a chain of connected partitions. The resulting tree-like decomposition provides a variable order for guiding CNF-SAT search.

Experiments carried out over a large and varied set of benchmarks demonstrate that:

- Our partitioning procedure is very fast and scalable - partitioning a large set of constraints in a matter of seconds.
- The variable order derived through the partitioning results in significant increase in performance (often orders of magnitude) of the SAT engine.
- Our approach is a viable alternative to contemporary minimum tree-width decomposition techniques in the context of deriving a good variable order for SAT search.

For our experiments, we use the state-of-the-art hyper-graph partitioning tool called HMETIS [42] [43] and port it towards our problem of interest. To search for SAT solutions, we use a modified version of the zCHAFF SAT solver [12]. The reason for using zCHAFF is primarily because of its source code being available in public domain. Note that CNF-based SAT and hyper-graph based partitioning methods are fundamental problems in VLSI-CAD. These problems have been well researched and well established; as such they are not a subject of this paper.

IV. Constraint Decomposition via Hypergraph Partitioning

The given SAT problem in standard DIMACS CNF formulae is converted into a hyper-graph, where variables (as opposed to literals) are represented as hyper-graph edges and clauses are modeled as vertices. A balanced min-cut bi-partitioning is applied using hMeTiS. Figure 4 depicts the resulting partitioned sets of constraints. The variables that connect both partitions are termed as **cut-set variables**. These variables correspond to clauses that appear in different partitions. The two created partitions are termed as *LEFT* and *RIGHT* partitions.

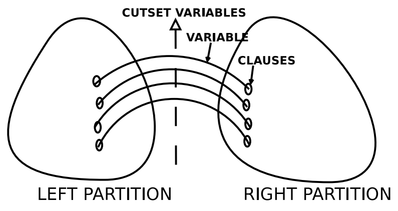

Fig. 4. Balanced bipartitioning

While analyzing the partitioning, we made an interesting observation related to the activity statistics of the cut-set variables. We observed that most of the variables forming the cut-set had high activity in the overall problem. Recall that the *activity* of a variable is defined as the frequency of occurrence of the variable among clauses. 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. It is important to begin the search by making assignments to variables of high activity [33] [12]. This implies that the cut-set variables should be the first choice for case-splitting. Therefore, these variables and corresponding constraints should form the first level of partition. To achieve this, the cut-set variables are extracted from both the partitions (LEFT and RIGHT) and are collected together to form a third partition as shown in Fig. 5. As a result, the extracted subproblem corresponds to clauses with high activity variables. Moreover, the LEFT and RIGHT partitions have no directly connecting hyper-edges.

One might be tempted to compare the above procedure to Amir's most constrained subproblem extraction [30]. However, there are subtle differences between the two. Their technique first decomposes the entire problem into loosely connected subproblems and then these subproblems are ordered/resolved in decreasing order of their clause-to-variable ratio. Their decomposition, though, is performed by analyzing clause connectivity alone. Our subproblem extraction also appears to be similar to that of the separator set extraction by Gupta *et al.* [32]. Their approach too extracts a set of constraints that contain high activity variables. However, from this point onwards, our approach deviates from theirs. They recursively perform this separator set extraction to produce smaller subproblems which are disjoint, so that BDD's can be built efficiently for Image Computation.

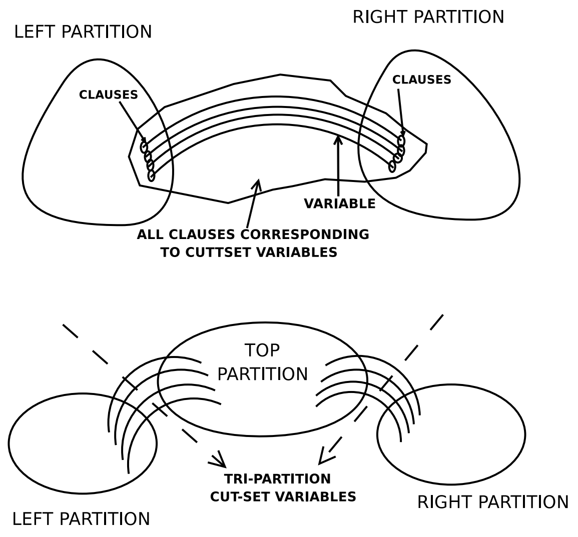

Fig. 5. Subproblem with high activity variables extracted from the bipartition

In contrast, we perform this subproblem extraction only once. Subsequently, we analyze the activity of the tri-partition cut-set variables (Fig. 5) to further decompose the problem. This is explained below.

A. Decomposition based on variable activity and clause connectivity

As shown in Fig. 5, a top-level partition is created by extracting the clauses corresponding to the cut-set variables. As a first step, these (bi-partition) cut-set variables are ordered according to their activity and stored in a list. The SAT tool will branch on these variables first. Note that, the clauses in top-level partition contain a set of variables, other than the cut-set, which would correspond to the first level of connectivity among constraints. This is shown in Fig. 6 as LEVEL-1 CONNECTIVITY. Subsequently, the clauses corresponding to the LEVEL-1 CONNECTIVITY VARIABLES are extracted to form the next level of partition. Again, these LEVEL-1 CONNECTIVITY VARIABLES are ordered according to their corresponding activity. To break ties, Level-1 connectivity variables are ordered according to their activity within the Level-1 partition. This subset of ordered variables is appended to the list. Repeating the above procedure, results in a fully decomposed tree as shown in Fig. 7.

The operation of our algorithm can be visualized for the example circuit show in Fig. 1. The decomposition tree is shown in Fig. 3. It can be observed from the circuit that the activity of variables {6, 7, 5, 8} is the highest. Variables {7, 5} form the bi-partition cut-set created by HMETIS. Clauses corresponding to these cut-set variables are extracted to form the top-level partition. Observe that variables {3, 6, 8, 1, 2} are the Level-1 connectivity variables. Ordering these Level-1 connectivity variables according to their activity results in {6, 8, 1, 2, 3}. Finally, variables {9, 4} result in the next level of connectivity. Hence, the order for SAT search is {7, 5, 6, 8, 1, 2, 3, 9, 4}.

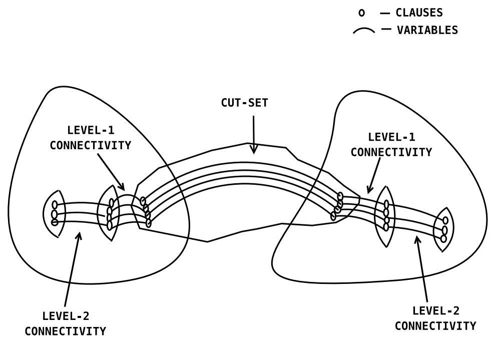

Fig. 6. Tree Decomposition

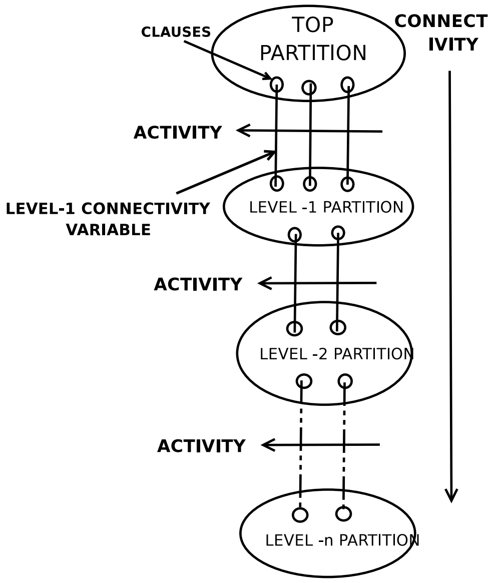

Fig. 7. Fully decomposed SAT problem

V. EXPERIMENTAL RESULTS AND ANALYSIS

The above approach has been programmed as an algorithm which is integrated with both HMETIS [42] and zCHAFF [12]. The algorithm first converts the CNF-SAT constraints to a hypergraph and invokes HMETIS to perform the balanced min-cut partitioning. The resulting partitions are analyzed to perform the decomposition and, subsequently, the required variable order is derived. This variable order is given to the CNF-SAT tool zCHAFF as an initial order on which branching/decision making is performed. On encountering conflicts during the search, we allow zCHAFF to add conflict induced clauses and proceed with its book-keeping and (non-chronological) backtracking procedures. Using this setup, experiments were conducted over a wide range of benchmarks from: (i) DIMACS suite [44]; (ii) Miter circuits; (iii) Fpga routing benchmarks; (iv) Urquhart problems [45]; (v) Micro-processor verification benchmarks [39]. The results are analyzed below.

First, we compare the time required to derive the variable order by our approach with that of Amir *et al.* [38] and MINCE [40]. Some results for the larger and harder-to-solve CNF-SAT instances are presented in the Table I. As it can be observed

TABLE I
COMPARISON OF OUR PARTITIONING TIME WITH ORIGINAL zCHAFF RUNTIME, AMIR'S AND MINCE'S PARTITIONING TIME

Bench-mark	Vars/ Clauses	zCHAFF (sec)	Amir (sec)	MINCE (sec)	Ours (sec)
c2670_opt	2527 / 6438	1.48	16.54	8.23	1.15
c3540_opt	3431 / 9262	20.57	23.66	13.95	1.42
c5315_opt	4992 / 14151	34.62	54.56	25.51	1.25
c7552_opt	5466 / 15150	105.97	71.30	24.43	1.76
4pipe	5237 / 80213	111.2	505.83	93.1	13.84
5pipe	9471 / 195452	167.22	>2000	267.2	38.40

from the table, 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, our approach can derive the variable order much faster than the other two. This clearly demonstrates the computational limitations of these methods; as such Amir's and MINCE approach are too expensive to be applicable for large CAD problems.

We now demonstrate that the variable order derived by our technique results in significant speed up (orders of magnitude in many cases) over the one conventionally used by zCHAFF. The experiments were performed on both satisfiable and unsatisfiable instances. For the satisfiable instances, we experimented with the FPGA routing benchmarks, which are known to be difficult-to-solve instances and for the unsatisfiable instances, we experimented with the Velev's benchmark suite (FVP UNSAT 1.0 , FVP UNSAT 2.0), Urquhart problems [45] and the miter circuits. The results are presented in Table II. Also, most of the benchmarks we experimented are both large and hard-to-solve instances. In the table, column 3, 4 and 5 corresponds to the original zCHAFF solving time, number of decisions and number of implications, respectively. Column 6 shows the tree decomposition time of our approach and column 7 shows the solving time of our modified zCHAFF solver. Column 8 shows the total time required to solve the problem, including both search time and partitioning time. Column 9 and 10 corresponds to the number of decisions and implications generated during our SAT search.

It is clearly seen from the table that the performance of CNF-SAT solver has been greatly improved with our variable order. The run times of our proposed approach are significantly smaller than that of original zCHAFF SAT solver, even for the larger and more difficult instances. Note that our approach performs better than original zCHAFF for most of these benchmarks - barring a few for which the compute times are comparable. It can be also noted that as compared to original zCHAFF, our results show consistent improvements in the number of decisions as well as implications made by zCHAFF using the decision order derived by our proposed method.

A. Advantages and Limitations of Our Approach

One of the most important advantages of our approach over that of the other tree decomposition ones is that our technique is scalable and hence, applicable to practical CAD problems. Even though, we do not perform minimum tree-width decomposition directly, the variable order derived by our approach sig-

TABLE II

RUN-TIME COMPARISON OF OUR PROPOSED APPROACH WITH zCHAFF

		Original zCHAFF			Modifi ed zCHAFF				
Bench-mark	Vars/ Clauses	Time (sec)	Deci-sions	Implica-tions	Partition Time(s)	Solve Time(s)	Total Time(s)	Deci-sions	Implica-tions
fpga10_8	120 / 448	15.66	54,199	980,436	0.16	0.01	**0.17**	460	3,935
fpga10_9	135 / 549	95.64	160,892	2,736,044	0.166	0.03	**0.196**	1,714	26,599
fpga12_8	144 / 560	244.42	279,070	5,749,638	0.173	0.41	**0.583**	5,674	95,031
fpga12_9	162 / 684	>1000	—	—	0.199	1.13	**1.329**	14,095	295,693
fpga12_11	180 / 820	>1000	—	—	0.278	3.46	**3.738**	26,521	433,912
fpga12_12	198 / 968	727.44	455,458	7,846,633	0.02	0.288	**0.3**	1,679	16,670
fpga13_10	195 / 905	>1000	—	—	1.14	0.231	**1.371**	12,949	219,829
fpga13_12	234 / 1242	>1000	—	—	0.06	0.353	**0.413**	2,250	32,698
Urq3_1	43 / 334	112.05	1,053,197	12,730,779	0.102	7.81	**7.912**	174,977	1,109,054
Urq3_4	36 / 220	**0.07**	6,098	38,021	0.083	0.08	0.163	4,837	45,992
Urq3_9	37 / 236	3.37	80,290	1,025,862	0.076	1.68	**1.756**	41,869	351,605
Urq3_10	37 / 236	3.37	80,290	1,025,862	0.076	0.93	**1.006**	25,012	173,403
c880_opt	770 / 2126	1.13	16,911	812,548	0.362	0.33	**0.392**	11,477	432,012
c1355_opt	1006 / 2954	13.62	98,931	8,560,559	0.549	0.46	**1.009**	14,659	631,135
c1908_opt	1895 / 5023	**1.67**	20,014	263,4902	0.9	0.82	1.72	14,053	1,508,262
c2670_opt	2527 / 6438	**1.48**	45,713	1,921,714	1.15	1.38	2.53	53,150	1,993,412
c3540_opt	3431 / 9262	**20.57**	74,325	16,600,030	1.42	22.22	23.64	83,304	18,984,371
c5315_opt	4992 / 14151	34.62	136,531	19,829,630	1.25	13.35	**14.6**	128,008	13,129,260
c7552_opt	5466 / 15150	105.97	384,776	42,128,278	1.76	39.7	**41.46**	257,514	25,418,457
3pipe	2468 / 27533	**1.67**	32,816	1,956,556	4.27	3.27	7.54	54,194	3,402,468
4pipe	5237 / 80213	111.2	471,592	71,488,318	13.84	74.27	**88.11**	415,351	53,194,998
5pipe	9471 / 195452	167.22	1,773,807	94,373,254	38.40	86.76	**125.16**	875,782	46,788,886
3pipe_k	2391 / 27405	**2.52**	48,902	2,796,441	3.526	2.4	5.926	43,815	2,478,836
4pipe_k	5095 / 79489	184.2	711,615	106,337,088	12.327	51.62	**63.947**	256,708	38,272,072
5pipe_k	9330 / 189109	764.47	1,823,949	417,340,698	39.43	384.57	**424.0**	1,337,479	240,184,009
3pipe_q0_k	2476 / 25181	8.6	123,615	5,294,597	4.02	1.51	**5.53**	40,940	2,009,529
4pipe_q0_k	5380 / 69072	56.83	394,973	51,775,049	13.133	24.59	**37.723**	227,948	26,294,080
5pipe_q0_k	10026 / 154409	573.25	15,72,754	374,623,438	37.231	395.39	**432.62**	1,754,215	444,278,100
9vliw_bp_mc.cnf	20093 / 179492	**235.21**	3,619,845	94,144,205	33.05	238.39	271.44	3,741,337	88,779,625

nificantly increases the performance of SAT search. Another advantageous application of our approach is that of efficient incremental SAT solving. As a set of constraints generated incrementally, our decomposition scheme can guide SAT diagnosis by suggesting the order in which the variables and clauses should be added to the system for efficient resolution.

As far as the limitations of our approach is concerned, our partitioning would not be very effective for instances such as the NQueens problem where all variables are present in all the clauses. Thus, the graph represents a clique. In such a case, when our top level partition is extracted it would encompass the entire set of constraints. However, it can be shown that minimum width tree decomposition techniques would also fail in those instances.

VI. CONCLUSIONS AND FUTURE WORK

This paper has presented a constraint partitioning scheme that can be exploited to derive a good variable order that can be used to guide CNF-SAT search for faster SAT solving. As opposed to contemporary minimum tree-width decomposition schemes, our approach performs constraint partitioning on a hypergraph by analyzing both variable activity and clause connectivity. As a result, not only is our technique fast, scalable and robust, it also generates a variable order that outperforms those generated by contemporary tools. A novel partitioning procedure has been proposed and its effectiveness has been demonstrated over a wide range of benchmarks of large size and high difficulty. Our approach has demonstrated that constraint partitioning based SAT methods can be made to work in practice as a viable alternative to minimum width tree decomposition schemes for SAT. As part of future work, we would like to analyze the effect of our partitioning with applications to both Incremental SAT and parallel SAT.

REFERENCES

[1] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[2] R. E. Bryant, "Graph Based Algorithms for Boolean Function Manipulation", *IEEE Trans. on Computers*, vol. C-35, pp. 677–691, August 1986.

[3] M. Davis and H. Putnam, "A Computing Procedure for Quantifi cation Theory", *Journal of the ACM*, vol. 7, pp. 201–215, 1960.

[4] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving", *in Communications of the ACM, 5:394-397*, 1962.

[5] C. E. Blair and *et al.*, "Some Results and Experiments in Programming Techniques for Propositional Logic", *Comp. and Oper. Res.*, vol. 13, pp. 633–645, 1986.

[6] T. Larabee, *Efficient Generation of Test Patterns using Satisfiability*, PhD thesis, Dept. of Computer Science, Stanford University, Feb. 1990.

[7] P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Combinational Test Generation using Satisfi ability", Technical Report UCB/ERL M92/112, Dept. of EECS., Univ. of California at Berkeley, Oct. 1992.

[8] J. W. Freeman, *Improvements to Propositional Satisfiability Search Algorithms*, PhD thesis, Dept. of Comp. and Inf. Sc., Univ. of Penn., May 1995.

[9] R. Zabih and D. A. McAllester, "A Rearrangement Search Strategy for

Determining Propositional Satisfiability", *in Proc. Natl. Conf. on AI*, 1988.

[10] J. Marques-Silva and K. A. Sakallah, "GRASP - A New Search Algorithm for Satisfiability", *in ICCAD'96*, pp. 220–227, Nov. 1996.

[11] H. Zhang, "SATO: An Efficient Propositional Prover", *in In Proc. Conference on Automated Deduction*, pp. 272–275, 1997.

[12] 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.

[13] E. Goldberg and Y. Novikov, "BerkMin: A Fast and Robust Sat-Solver", *in DATE, pp 142-149*, 2002.

[14] W. Kunz and D.K. Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems – Test, Verification and Optimization", *IEEE Tr. on CAD*, vol. 13, pp. 1143–1158, Sep. 1994.

[15] K. S. Brace, R. Rudell, and R. E. Bryant, "Efficient Implementation of the BDD Package", *in DAC*, pp. 40–45, 1990.

[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] R. Damiano and J. Kukula, "Checking Satisfiability of a Conjunction of BDDs", *in DAC, pp 818-823*, 2003.

[18] 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.

[19] 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.

[20] 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.

[21] M. Prasad, P. Chong, and K. Keutzer, "Why is ATPG Easy?", *in Design automation Conf.*, pp. 22–28, June 1999.

[22] 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.

[23] F. Aloul and *et al.*, "Satometer: How Much Have We Searched?", *IEEE Trans. CAD*, vol. 22, pp. 995–1004, 2003.

[24] 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.

[25] E. Goldberg and Y. Novikov, "Verification of Proofs of Unsatisfiability for CNF Formulas", *in Design, Automation and Test in Europe (DATE)*, 2003.

[26] S. Pilarski and G. Hu, "SAT with Partial Clauses and Backleaps", *in In Proc. DAC*, pp. 743–746, 2002.

[27] R. Dechter and J. Pearl, "Network-based Heuristics for Constraint-Satisfaction Problems", *Artificial Intelligence*, vol. 34, pp. 1–38, 1987.

[28] R. Dechter, *Constraint Processing*, chapter 9, Morgan Kaufmann Publishers, 2003.

[29] E. Amir and S. McIlraith, "Partition-Based Logical Reasoning", *in 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'2000)*, 2000.

[30] 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.

[31] 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.

[32] 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.

[33] J. P. M. Silva, "The Impact of Branching Heuristics in Propositional Satisfiability Algorithms", *in Portuguese Conf. on Artificial Intelligence*, 1999.

[34] E. Amir, "Efficient Approximation for Triangulation of Minimum Treewidth", *in 17th Conference on Uncertainty in Artificial Intelligence (UAI '01)*, 2001.

[35] A. Caldwell, A. Kahng, and I. Markov, "Improved Algorithms for Hypergraph Bipartitioning", *in In Proc. Asia-Pacific DAC*, 2000.

[36] A. Darwiche, "Decomposable Negation Normal Form", *J. ACM*, vol. 48, pp. 608–647, 2001.

[37] A. Darwiche, "Compiling Knowledge into Decomposable Negation Normal Form", *in Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[38] E. Amir and S. McIlraith, "The Partitioning and Reasoning Project", http://www-faculty.cs.uiuc.edu/ eyal/decomp/.

[39] M. Velev, "Sat benchmarks for high-level microprocessor validation", http://www.cs.cmu.edu/ mvelev.

[40] F. Aloul, I. Markov, and K. Sakallah, "MINCE", http://www.eecs.umich.edu/ faloul/Tools/mince/.

[41] S. Arnborg, D. G. Corneil, and A. Proskurowski, "Complexity of Finding Embeddings in a k-tree.", *SIAM Journal of Algebraic and Discrete Methods*, vol. 8, 1987.

[42] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Application in VLSI Do main", *in Proc. DAC*, pp. 526–529, 1997.

[43] G. Karypis and V. Kumar, "Multilevel k-way Hypergraph Partitioning", *in Proc. DAC*, pp. 343–348, 1999.

[44] DIMACS, "Benchmarks for boolean sat", ftp://dimacs.rutgers.edu/pub/challange/sat/benchmarks.

[45] A. Urquhart, "Hard Examples for Resolution", *J. ACM*, vol. 34, pp. 209–219, 1987.