Guiding CNF-SAT Search by Analyzing Constraint-Variable Dependencies and Clause Lengths

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: *The type of decision strategies employed for CNF-SAT have a profound effect on the efficiency and performance of SAT engines. Over the years, a variety of decision heuristics have been proposed; each has its own achievements and limitations. This paper re-visits the issue of decision heuristics and engineers a new approach that takes an integrated view of the overall problem structure. Our approach qualitatively analyzes clause-variable dependencies by accounting for variable/literal activity, clause connectivity, distribution of variables among clauses of different lengths, and correlation among variables, to derive an initial static ordering for SAT search. To account for conflict clauses and their resolution, a corresponding dynamic variable order update strategy is also presented. Quantitative metrics are proposed that are used to devise an algorithmic approach to guide overall SAT search. Experimental results demonstrate that our strategy significantly outperforms conventional approaches.*

I. Introduction

Recent advances in SAT have significantly impacted Electronic Design Automation (EDA) applications such as functional verification, logic synthesis, model checking, equivalence checking, etc. Much of these advances can be credited to efficient branching heuristics and conflict resolution procedures that have been researched over the years. The order in which variables (and correspondingly, constraints) are resolved significantly impacts the performance of SAT search procedures [1] [2].

Early branching heuristics, such as Bohm's heuristic [3], Maximum Occurrences on Minimum sized clauses (MOM) [4] and Jeroslow-Wang heuristic [5], attempt to resolve smaller clauses first as that might result in earlier conflicts and implications. However, these heuristics are unable to solve large (industrial) problems that one encounters in contemporary EDA problem formulations. Subsequently, a new class of decision heuristics, such as DLIS [6] and VSIDS [1], were derived which utilize the *counts of variables or literals* appearing in the clauses. Activity of a variable (or literal) - which is defined as its frequency of occurrence among clauses - plays an important role in such heuristics. The DLIS heuristic branches on the literal which has the highest activity among unsatisfied clauses. On the other hand, the VSIDS heuristic associates a score for each literal and branches on the literal with the highest score. This score is initially set to the literal's activity among all clauses. The score is updated whenever conflict clauses are added. Also, the heuristic divides these scores in a periodic manner to avoid overflow and to give importance to variables that appear in recent conflict clauses. Most conventional SAT solvers [1] [2] [6] employ variations of the such branching heuristics to resolve the constraints.

In recent past, a lot of effort has been invested in deriving variable orderings for SAT search by analyzing the problem structure. In particular, connectivity of constraints has been used as a means to efficiently model and analyze the clause-variable dependencies. Clause connectivity can be modeled by representing CNF-SAT constraints on a (hyper-) graph. Subsequently, analyzing the graph's topological structure allows to derive an "ordering of variables" that is used to guide the search.

Tree decomposition techniques have been proposed in literature [7] [8] 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 [8] [9] [10] [11] [12] [13]. Various approaches operate on such partitioned tree structures by deriving an order in which the partitioned set of constraints are resolved [9] [10] [11] [13]. MINCE [14] employs CAPO placer's mechanism [15] to find a variable order such that the clauses are resolved according to their chain of connectivity. Bjesse *et. al.* [10] proposed tree decomposition based approaches to guide variable selection and conflict clause generation. Aloul *et. al.* have proposed a fast, heuristic based approach, FORCE [16], as an alternative to the computationally complex approach of MINCE. Durairaj *et. al.* [17] proposed a hypergraph bi-partitioning based constraint decomposition scheme that simultaneously analyzes variable-activity and clause-connectivity to derive a variable order for SAT search. They have further proposed a variable ordering scheme (ACCORD [18]) that analyzes correlations among pairs of variables to resolve the constraints.

The above variable ordering schemes are generally employed to derive an *initial static variable order.* SAT tools use this order and begin the search. SAT solvers update this variable order dynamically when conflict clauses are added. Due to this update, the original ordering gets destroyed/modified, over a period of time. As a result, the "speed-up potential" of the initial variable order is not fully exploited. Moreover, some of these ordering schemes suffer from long compute times [7] [14] [9]. Furthermore, both the initial variable ordering schemes, as well as contemporary decision heuristics (DLIS, VSIDS), fail to explicitly account for the effect of clause lengths on decision making.

Our Approach: Our approach first computes an initial variable order for SAT search by analyzing, both qualitatively and quantitatively, the overall problem structure. Following are the salient features of our initial ordering scheme. To analyze the distribution of variables among clauses, we develop a quantitative metric that combines variable/literal activity scores along with the length of clauses in which they appear. Our approach also quantitatively analyzes how tightly the variables are related to each other. Moreover, this variable correlation is further scaled with respect to the length of clauses. Our heuristic also considers the effect of decision assignments on subsequent constraint resolution.

We begin the search using this initial variable order. As and when conflicts are encountered, and corresponding conflict clauses are added to the database, the variable order has to be updated so as to take into account the increase in variable activity as well as conflict clause lengths. For this purpose, we propose a dynamic strategy that correspondingly updates the variable order.

While some of these concepts have been presented in literature before, ours is an attempt to take an integrated view of the problem structure. Experimental results demonstrate that our *hybrid* approach is faster and much more robust than contemporary techniques. Significant improvements (in some cases, an order of magnitude) are observed over a wide range of EDA as well as non-EDA applications.

This paper is organized as follows. Section II reviews previous decision heuristics. Section III proposes a new hybrid score for literals that accounts for both literal activity and clause lengths. Section IV presents our initial variable ordering scheme. Section V discusses a strategy to update this order during the search. The overall approach is presented in Section VI and experiments are described in Section VII. Section VIII concludes the paper.

II. Literal Activity and Clause Length

While using literal counts as a metric to resolve the constraints is the norm now-a-days, it suffers from the following limitation. Activity based heuristics give equal importance to variables (literals) irrespective of the size of the clauses in which they appear. However, it can be noted that branching on a variable appearing in shorter clauses (say, a 2 literal clause) may produce faster/earlier implications or conflicts than for larger clauses. Hence, it is important to consider the effect of clause length while making decisions.

Most of the early branching heuristics [3] [4] [5] did consider clause length as an important metric in deciding the next branching variable. Let us revisit and analyze one of these heuristics, the Jeroslow-Wang (JW) heuristic. JW heuristic branches on a literal L that maximizes the metric

$$J(L) = \sum_{i, L \in C_i} 2^{-n_i} \tag{1}$$

over all literals L, where n_i is the number of literals in clause C_i. The motivation behind this heuristic is that in a list of clauses with n variables, there are 2^n truth valuations and a clause of length p rules out exactly 2^{n-p} of these valuations. Using the above motivation, Jeroslow-Wang justify their rule as one that tends to branch to a sub-problem that is most likely to be satisfiable ([5], pp. 172-173). However, [19] has experimentally disproved their original justification and shown that the reason behind the success of JW heuristic is that it creates simpler sub-problems. This, in turn, leads to faster SAT solving. Recently, Pilarski *et. al.* [20] have also shown that considering clause length while deciding on next branching variable produces significantly faster results for a certain classes of benchmarks. However, note that 2^{-n_i} is an exponentially decreasing function. Therefore, for large sized clauses, it generates very small numerical scores. As a result, it cannot properly differentiate between the effect of decisions on variables that appear in very large clauses. Such large clauses are commonly found in SAT problems, particularly in conflict clauses [20]. Hence, a better metric that properly accounts for clause lengths is required.

III. JW-Activity: A Hybrid Score for Literals

We propose a new quantitative metric that combines the activity together with clause lengths (JW heuristic). The above mentioned limitation (exponentially decreasing scores) of J(L) metric can be overcome by up-scaling the scores. We scale the J(L) metric using the activity of literal L. In other words,

$$JW - Activity(L) = activity(L) * J(L) \tag{2}$$

For computing JW-Activity, we have modified the data structure of the conventional solver to store information regarding clause lengths. For this purpose, an array is included in the database whose purpose is store each literal's score corresponding to JW heuristic (equation 1). Hence, whenever a clause is initially read into the database, the JW score for each literal is updated. Along with this score, the solver is also allowed to update its literal activity. The JW-Activity for each literal is thus computed.

Note that when the search proceeds, some clauses will be satisfied. This requires to update the JW score dynamically. However, this update is generally very expensive as all satisfied clauses have to be recognized; thus breaking the two-literal watching scheme [1]. This issue is obviated in our overall approach - a discussion on which follows in Section VI.

IV. Incorporating Connectivity-based Decision Heuristics

The importance of deriving a good variable ordering to guide SAT search is well known, as discussed in [21] [14] [22]. Activity based scores alone are not sufficient to derive a good initial variable order. Therefore, contemporary techniques analyze the variable activity along with clause connectivity and derive an initial static variable order. SAT tools use this order for decision assignments. In the previous section, we have introduced a new hybrid score to model literal activity along with clause lengths. In addition to JW-Activity, we now wish to analyze the connectivity information within the problem structure, that too with respect to clause lengths, and derive a similar initial variable ordering.

A recent approach, ACCORD [18], exploits the activity-connectivity information to derive an initial variable order. It has been experimentally demonstrated to be faster and more robust than other connectivity based approaches, such as [14] [9] [16] [17]. We exploit and extend the main concepts of ACCORD to derive a variable order that utilizes problem structure and clause length. This approach will form the basis of our initial variable order derivation scheme.

One of the interesting features of ACCORD is the metric that measures how tightly the variables are related to each other. This metric is defined as follows:

Definition IV.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**.

ACCORD models variable activity, connectivity and the correlation of variables on a graph and subsequently analyzes its topology to derive a variable order [18]. While ACCORD did show improvements on non-EDA problems, its performance was not satisfactory on larger EDA benchmarks (such as the pipeline verification benchmarks) [18]. We analyzed the algorithm and found that even though ACCORD tightly analyzes the constraint-variable dependencies, it does not incorporate the effect of clause length when deciding upon the next branching variable. For example, consider the following set of clauses:

$$(x + a + e + b + f + g)(x + c + y)$$

According to ACCORD, the degree of correlation between the variable x and every other variable in the above set of constraints is measured to be same. However, it can be noted that by resolving the three literal clause first will produce earlier implications/conflicts.

A. Enhancing ACCORD: JW-ACCORD

To overcome the above mentioned limitation, we decided to incorporate the effect of clause length within ACCORD. We extend the concept of degree of correlation by incorporating the clause length information within the metric. The degree of correlation between a pair of variables $(x_i,\ x_j)$ is normalized according to the length of the clauses in which they appear together. For example, suppose that the pair of variables $(x_i,\ x_j)$ appear together in two clauses C1 and C2. Let C1 be a 3 literal clause and C2 be a 4 literal clause. Then the corresponding normalized degree of correlation is equal to $2^{-3} + 2^{-4}$. Formally stating, the normalized degree of correlation between a pair of variables $(x_i,\ x_j)$ is computed as

$$\sum_{l, X_i \& X_j \in C_l} 2^{-n_l}$$

where x_i,x_j appear together in clauses C_l and n_l is the number of literals appearing in each C_l.

Using the new metric, we now describe the modified JW-ACCORD algorithm by means of an example. Let us consider the CNF-SAT problem shown below.

$$(x + a)(x + b + c + d)(\bar{a} + y + z)$$
$$(\bar{x} + \bar{y})(\bar{a} + \bar{z})(\bar{x} + \bar{b} + \bar{c} + \bar{d} + \bar{e})$$

TABLE I
JW-ACCORD: Various metrics for the above CNF formulae

Literal(s)	Literal Activity	J(L)	JW-Activity
x	2	$2^{-2} + 2^{-4}$	0.625
$\bar{x}$	2	$2^{-2} + 2^{-5}$	0.5625
a	1	2^{-2}	0.25
$\bar{a}$	2	$2^{-2} + 2^{-3}$	0.75
y, z	2	2^{-3}	0.125
$\bar{y}, \bar{z}$	1	2^{-2}	0.25
b, c, d	1	2^{-4}	0.0625
$\bar{b}, \bar{c}, \bar{d}, \bar{e}$	1	2^{-5}	0.03125

Table I presents various metrics computed for the above set of constraints. Column 1 in the table corresponds to the literal for which the metric has been computed. Column 2 and 3 shows the computed literal activity and the J(L) metric (equation 1) respectively. Column 4 corresponds to the JW-Activity score, which is computed by multiplying values of column 2 with those in column 3.

The constraint-variable relationships for the given CNF-SAT problem can be modeled as a weighted graph as shown in Fig. 1. The variables form the vertices, while

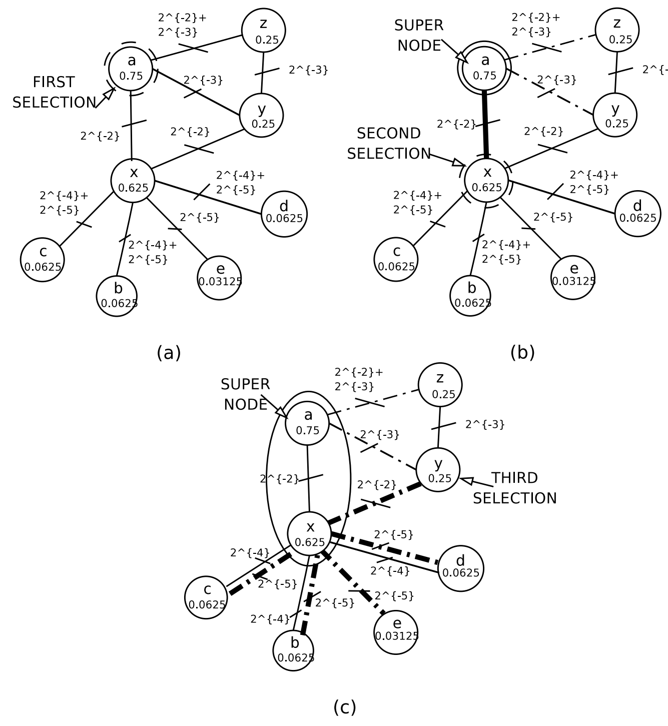

Fig. 1. JW-ACCORD : Edge weights denote the normalized degree of correlation between the variables (vertices). JW-Activity score depicted within the node. Edges between b, c, d and e are not shown.

edges denote the correlation/connectivity between them. Associated with each variable is JW-Activity measure corresponding to the literal that has the higher score. For example, in Fig. 1 (a), the value of 0.75 in node a corresponds to that of literal $\bar{a}$ as that has the higher score than literal a (see Table I). The edge weights represent the normalized degree of correlation between the variables/vertices. For example, variables x and c appear together in 2 clauses - in the second and the last clause. The length of the second clause is equal to 4 and that of the last clause is 5; hence, the edge weight is equal to $2^{-4} + 2^{-5}$. An ordering of the nodes (variable order) is performed by analyzing the graph's topology.

JW-ACCORD begins the search with the literal that has the highest value for JW-Activity metric. Here, from the table I, it can be seen that literal $\bar{a}$ has the highest JW-Activity metric (i.e. 0.75). In contrast, ACCORD would have considered the literals $\{x, \bar{x}, \bar{a}\}$ to begin the search, as they share the same literal activity. Therefore, this variable a is marked and the node is added to a set called *supernode* and also stored in a list (*var_ord_list*). Next we need to identify the set of variables connected to this supernode a. Note that, when the solver branches on this literal $\bar{a}$, it will assign $a = 0$. Hence, all the clauses corresponding to the literal $\bar{a}$ will be satisfied due to this assignment. In order to exploit this behaviour, the algorithm determines connectivity only from the clauses in which literal a appears. In that case, only the variable x is *functionally* connected to the supernode. Hence, x is marked as the next branching literal and it is added to the supernode as shown in Figs. 1 (b) and (c). Now we need to find the next set of variables that are connected to the current supernode $\{a, x\}$. Again, in order to utilize the effect of decision making, only clauses containing $\bar{x}$ and a are considered for connectivity analysis. For this purpose, the edges incident on the node x are split into two, corresponding to the (positive and negative) literals and their correlation. This is shown in Fig. 1 (c). The dotted edge corresponds to the negative literal ($\bar{x}$). For example, between node x and y, there is only a dotted edge as y appears together with $\bar{x}$ and not with x. Continuing with our procedure, the variables that are connected to $\bar{x}$ and a are $\{y, b, c, d, e\}$. The correlation between $\bar{x}$ and node y is the highest and hence, y is the next branching variable. Actually, since $\bar{y}$ has higher score than y, $\bar{y}$ is the next branching literal (i.e. $y = 0$ will be the next assignment). The algorithm continues by analyzing such normalized degrees of correlation among literals until all the nodes are ordered. In case of a tie, the literal with higher JW-Activity score is selected.

Note, the above is just a visualization of our approach. In practice, we do not explicitly construct this weighted graph. The correlations are analyzed by operating directly on an array of array data structure as shown in Algorithm 1.

Algorithm 1 Pseudo code for JW-ACCORD

```
1:  INPUT = CNF Clauses and Variable-Clause Database ;
2:  /* Variable-Clause database is implemented as array of arrays */
3:  /* For each variable $x_i$, the Variable-Clause database contains
    a list of clauses in which $x_i$ appears */
4:  OUTPUT = Variable order for SAT search ;
5:  activity_list = Array containing activity of each literal ;
6:  JW_list = Array containing JW metric J(L) for each literal ;
7:  var_order_list = Initialize variable order according to scaled
    JW-Activity metric (activity_list[L] * JW_list[L]) ;
8:  connectivity_list = Initialize to zero for all variables ;
9:  int i = 0;
10: **while** i != number of variables **do**
11:   /* Implicitly, supernode = {var_order_list[0], ..., var_order_list[i]} */
12:   next_var = var_order_list[i] ;
13:   next_lit = The positive or negative literal of next_var that has
      the highest scaled JW-Activity score; /* Effect of Decision assignments */
14:   correlation_list = find variables connected to next_lit using
      Variable-Clause database along with their clause length;
15:   **for all** var ∈ correlation_list **do**
16:     connectivity_list[var] += ($2^{-clauselength}$ * activity of next_lit);
17:     adjust_variable_order(var) ;
18:     /* Linear sort is used to update the variable order corresponding
        to both connectivity_list as well as activity_list */
19:   **end for**
20:   i = i+1;
21:   /* At this point, {var_order_list[0], ..., var_order_list[i]} is the
      current variable order*/
22: **end while**
23: return(var_order_list) ;
```

A weak upper bound for the JW-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 JW-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.

V. Accounting for Conflict clauses

The above presented algorithm JW-ACCORD is used to derive an initial static variable order. The SAT tool will use this order for decision assignments. As the search proceeds, conflicts may be encountered and conflict clauses would be added to the database. As and when conflict clauses are added, the scores of literals have to be updated. We update the JW-Activity score as follows: (i) J(L) metric for each literal appearing in the conflict clause is updated according to the conflict clause length; (ii) activity of these literals is incremented by one; (iii) JW-Activity is then computed from this new activity and J(L) metric. According to the new JW-Activity score, the variable order is updated.

In a recent approach (BerkMin) [2], it was demonstrated that information within the conflict clauses (particularly the more recent conflicts) needs to be analyzed for branching. If there are unresolved conflict clauses, BerkMin selects the highest active literal within the conflict clauses for immediate branching. However, BerkMin does not analyze clause lengths. It is important to do so because conflict clause size often varies from 2 literals to 100's of literals [2] [20]. Keeping this in mind, we also keep track of JW-Activity scores of literals appearing within conflict clauses using a separate counter. Therefore, if there are unresolved conflict clauses, we resolve them first using the local JW-Activity score corresponding to the conflict clauses. Once all conflict clauses are resolved, we revert to the overall JW-Activity score to branch on the next undecided variable.

Our initial variable order can be used by any SAT solver and its native decision heuristics can certainly update the order (say, the VSIDS scheme in CHAFF [1]). However, that would modify our initial order unfavourably. We have performed experiments with both the static order and with our dynamic update strategy. Some results are shown in Fig. 2. From the figure, the importance of our dynamic updates is clearly visible.

VI. Overall Branching Strategy

Combining the above concepts, our overall strategy is as follows: When the problem is being read, we compute the J(L) scores as well as the literal activities. Subsequently, JW-Activity score is computed using which the literals are sorted in decreasing order. Then, our JW-ACCORD algorithm is applied to derive an initial static variable ordering. This ordering is used by the SAT engine to start resolving the constraints. As and when conflict clauses are added, the JW-Activity scores are modified accordingly and the variable order gets dynamically updated; allowing for non-chronological backtracks. The conflict clause resolution is also performed as described above (BerkMin-type strategy). We have implemented these procedures within the zCHAFF solver [1] (latest version developed in 2004) using its native data-structures.

When the search proceeds, we do not update the JW-score (and hence, the JW-Activity) when clauses become satisfied. Instead, we update the JW-Activity of literals only when new conflict clauses are added.

VII. Experimental Results and Analysis

We have conducted experiments over a large set of benchmarks that include: i) Microprocessor verification benchmarks [23]; ii) DIMACS suite [24]; iii) SAT encoding of Constraint Satisfaction Problems (CSP) [25]; iv) miter circuits submitted for SAT competition and v) hard instances specifically created for the SAT competition (industrial and handmade categories). We conducted our experiments on a Linux workstation with a 2.6GHz Pentium-IV processor and 512MB RAM.

Table II presents some results that compares the performance of the proposed decision heuristic with those of zCHAFF and original ACCORD variable ordering scheme [18]. Since, our algorithm is implemented within zCHAFF, for a fair comparison, zCHAFF is used as the base SAT solver for all experiments. Also, in order to compare the performance of our solver with other state-of-the-art solvers, we also ran experiments with MiniSAT SAT solver (latest version that participated in the SAT 2005 competition). Each benchmark is given to both MiniSAT and zCHAFF and their corresponding solving times are recorded, as reported in column 4 and 5. The benchmarks are then given to ACCORD to derive the variable order. The order derivation time is reported in column 6. This order is then given to zCHAFF as an initial static ordering. zCHAFF's native VSIDS heuristics update this order on encountering conflicts. This solve time is given in column 7 and column 8 gives total compute time.

Finally, our engineered tool is used to compute the JW-Activity scores and derive the initial order using JW-ACCORD; this time is reported in column 9. Note that the variable order time is negligible even for large benchmarks. Our modified tool uses this as the initial order and begins the search. On encountering conflicts, it employs the above described update strategy. This solve time is reported in column 10 and the total time in the last column. As compared to zCHAFF and ACCORD, the proposed heuristic results in significant speedup in SAT solving. In fact, our integrated approach *always defeats zCHAFF*. Our technique outperforms MiniSAT too, for most of the benchmarks - barring a few for which the compute times are comparable. Also, it can be noted

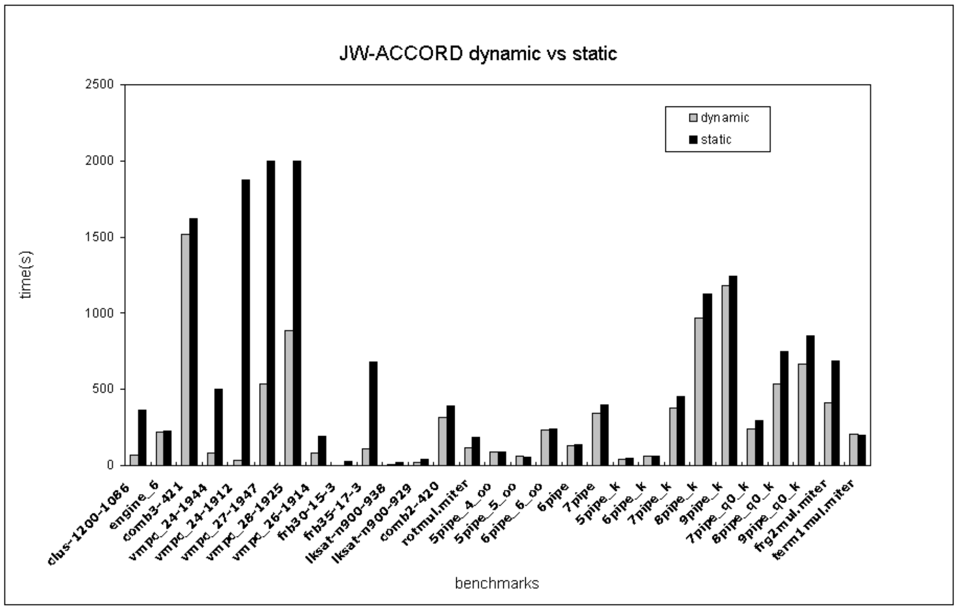

Fig. 2. Comparison of JW-ACCORD with static order only and with dynamic updates. Y-axis represents run time.

from the results that our solver can solve some benchmarks which none of the other tools can solve.

The experimental results does not include benchmarks that are: (i) trivially solvable (less than 10 seconds) by SAT solvers, such as smaller holes and smaller DIMACS benchmarks; (ii) unsolvable (within 2000 seconds) by any of the solvers including the proposed technique, such as larger pipeline verification benchmarks (10pipe, 11pipe).

VIII. Conclusions

This paper has engineered an initial variable ordering strategy along with corresponding variable order updates for CNF-SAT resolution. Clause-variable dependencies are analyzed along with clause lengths for this purpose. Not only does our approach qualitatively analyze how the variables are distributed among clauses of different sizes, but it also proposes a quantitative metric that models this behaviour. In order to analyze how tightly the variables are related to each other, the degree of correlation among pairs of variables is further normalized according to lengths of clauses in which they appear. Our approach is fast, robust, scalable and can efficiently handle a large set of variables and constraints. The proposed decision heuristic improves the performance of the solver by one or more orders of magnitude over a wide range of benchmarks.

References

[1] M. Moskewicz, C. Madigan, L. Zhao, and S. Malik, "CHAFF: Engineering and Efficient SAT Solver", *in Proc. Design Automation Conference*, pp. 530–535, June 2001.

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

[3] M. Buro and H. Kleine, "Report on a sat competition", Technical Report, University of Paderborn, November 1992.

[4] J. W. Freeman, *Improvements to propositional satisfiability search algorithms*, PhD thesis, University of Pennsylvania, 1995.

[5] R.G. Jeroslow and J. Wang, "Solving propositional satisfiability problems", *Annals of mathematics and Artificial Intelligence*, vol. 1, pp. 167–187, 1990.

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

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

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

[9] 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.

[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] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] V. Durairaj and P. Kalla, "Guiding CNF-SAT Search via

TABLE II
Run-time Comparison of zCHAFF, ACCORD-zCHAFF and JW-ACCORD-zCHAFF

			MiniSAT	zCHAFF	ACCORD-zCHAFF			JW-ACCORD-zCHAFF		
Bench- mark	Vars/ Clauses	SAT/ UNSAT	Solve (sec)	Solve (sec)	Var. Time(s)	Solve Time(s)	Total Time(s)	Var. Time(s)	Solve Time(s)	Total Time(s)
clus-1200-1085	1200/4800	S	**17.206**	>2000	0.01	>2000	>2000	0.003	698.335	698.338
clus-1200-1086	1200/4800	S	**4.527**	868.905	0.02	425.698	425.718	0.003	68.2	68.203
lksat-n900-938	900/3357	S	12.989	77.3762	0.02	4.79227	**4.8122**	0.002	6.149	6.151
lksat-n900-929	900/6174	S	29.175	39.611	0.02	2.68659	**2.7065**	0.003	18.877	18.880
urqh2x5-1473	53/432	U	**134.74**	1167.15	0	612.96	612.96	0.000	308.001	308.001
hanoi5	1931/14468	S	**36.745**	115.547	0.04	296.203	296.243	0.018	49.953	49.971
hanoi6-fast	7086/78492	S	**5.517**	450.819	0.27	296.307	296.577	0.072	212.221	212.293
hole12	156/949	U	>2000	482.432	0	449.634	449.634	0.000	226.482	**226.482**
3bitadd-31	8432/31310	S	>2000	96.6333	0.17	32.814	32.984	0.024	8.747	**8.771**
frb30-15-3	450/19084	S	11.098	13.093	0	14.6358	14.6358	0.001	3.054	**3.055**
frb35-17-3	595/29707	S	**11.158**	442.893	0	35.7266	35.7266	0.002	110.005	110.007
comb2-420	31933/112462	U	338.653	415.699	6.78	380.58	387.36	0.122	317.003	**317.125**
comb3-421	4774/16331	U	**132.931**	>2000	0.16	>2000	>2000	0.016	1523.574	1523.59
frg2mul.miter	10316/62943	U	444.591	826.61	1.01	878.141	879.151	0.043	410.702	**410.745**
rotmul.miter	5980/35229	U	424.847	223.161	0.34	192.946	193.286	0.024	114.264	**114.288**
term1mul.miter	3504/22229	U	>2000	487.108	0.15	806.837	806.987	0.014	205.365	**205.379**
vmpc-21-1923	441/45339	S	86.719	194.013	0.05	80.9987	81.0487	0.009	5.913	**5.922**
vmpc-22-1956	484/52184	S	22.038	54.3337	0.04	11.8632	11.9032	0.012	4.4363	**4.4483**
vmpc-24-1912	576/67872	S	416.418	>2000	0.05	>2000	>2000	0.014	36.4655	**36.4795**
vmpc-24-1944	576/67872	S	**57.542**	>2000	0.06	639.878	639.938	0.017	81.295	81.312
vmpc-26-1914	676/86424	S	225.098	191.798	0.06	134.228	134.288	0.018	79.106	**79.124**
vmpc-27-1947	729/96849	S	697.058	651.221	0.09	1458.96	1459.05	0.026	537.453	**537.479**
vmpc-28-1925	784/108080	S	>2000	>2000	0.08	>2000	>2000	0.023	882.883	**882.906**
engine-4	6944/66654	U	**17.866**	58.978	0.75	66.4139	67.1639	0.108	36.012	36.120
engine-4nd	7000/67586	U	**88.257**	931.91	0.74	690.694	691.434	0.110	231.156	231.266
engine-6	45303/606068	U	**66.882**	285.44	38.77	337.326	376.096	1.845	215.882	217.727
9vliw-bp-mc	20093/179492	U	**58.589**	72.0121	5.5	78.0281	83.5281	0.221	67.317	67.538
5pipe-4-oo	9764/221405	U	>2000	110.381	2.88	108.704	111.584	0.355	90.344	**90.699**
5pipe-5-oo	10113/240892	U	513.492	64.6082	2.79	60.0999	62.8899	0.427	58.479	**58.906**
6pipe-6-oo	17064/545612	U	396.031	303.408	12.61	341.066	353.676	1.334	231.635	**232.969**
6pipe	15800/394739	U	>2000	172.502	8.08	150.806	158.886	0.740	130.89	**131.63**
7pipe	23910/751118	U	>2000	534.536	23.4	581.154	604.554	1.694	340.895	**342.589**
5pipe-k	9330/189109	U	>2000	52.844	2.61	49.2705	51.8805	0.295	42.854	**43.149**
6pipe-k	15346/408792	U	>2000	92.5619	9.2	75.0716	84.2716	0.771	58.536	**59.307**
7pipe-k	23909/751116	U	>2000	530.551	21.91	588.298	610.208	1.649	374.776	**376.425**
8pipe-k	35065/1332773	U	>2000	1664.47	63.23	1547.94	1611.17	3.517	970.227	**973.744**
9pipe-k	49112/2317839	U	>2000	>2000	303.9	>2000	>2000	7.201	1178.339	**1185.54**
7pipe-q0-k	26512/536414	U	>2000	370.813	12.06	369.797	381.857	1.107	235.445	**236.552**
8pipe-q0-k	39434/887706	U	>2000	1070.92	27.37	951.637	979.007	2.087	530.699	**532.786**
9pipe-q0-k	55996/1468197	U	>2000	1435.93	11.26	1344.06	1355.32	4.004	659.687	**663.691**
9dlx-vliw-iq1	24604/261473	U	>2000	514.127	15.69	381.09	396.78	0.379	225.574	**225.953**
9dlx-vliw-iq2	44095/542253	U	>2000	>2000	42.09	>2000	>2000	0.877	625.677	**625.554**
9dlx-vliw-iq3	69789/968295	U	>2000	>2000	147.04	>2000	>2000	1.744	1261.676	**1263.42**

Efficient Constraint Partitioning", *in ICCAD*, pp. 498 – 501, 2004.

[18] V. Durairaj and P. Kalla, "Variable ordering for efficient sat search by analyzing constraint-variable dependencies", in F. Bacchus and T. Walsh, editors, *Theory and Applications of Satisfiability Testing: 8th International Conference, SAT 2005*, vol. 3569, pp. 415–422, 2005.

[19] J. Hooker and V. Vinay, "Branching rules for satisfiability", *Journal of Automated Reasoning*, vol. 15, pp. 359–383, 1995.

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

[21] M. R. Prasad, A. Biere, and A. Gupta, "A Survey of Recent Advances in SAT-based Formal Verification", *Intl. Journal on Software Tools for Technology Transfer (STTT)*, vol. 7, 2005.

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

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

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

[25] "SAT Encodings of Constraint Satisfaction Problems", http://www.nlsde.buaa.edu.cn/ kexu/benchmarks/benchmarks.htm.