Exploiting Hypergraph Partitioning for Efficient Boolean Satisfiability

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

ABSTRACT

This paper presents hypergraph partitioning based constraint decomposition procedures to guide Boolean Satisfiability search. Variable-constraint relationships are modeled on a hypergraph and partitioning based techniques are employed to decompose the constraints. Subsequently, the decomposition is analyzed to solve the CNF-SAT problem efficiently.

An important aspect of CNF-SAT search procedures is to derive an ordering of variables to guide constraint resolution. Most conventional SAT solvers [1] [2] [3] employ variable-activity based branching heuristics to resolve the constraints. Recently, tree-decomposition techniques, borrowed from constraint satisfaction problems, have been employed to derive variable orderings to guide SAT diagnosis. Even though these techniques provide good variable orders for some SAT instances, their computational complexity makes them impractical for solving large and hard CNF-SAT problems. To overcome this limitation, this research advocates the use of hypergraph partitioning methods to decompose the constraints. This decomposition suggests a good variable order for SAT search.

The contributions of this research are *two-fold*: 1) to engineer a constraint decomposition technique using hypergraph partitioning; 2) to engineer a constraint resolution method based on this decomposition. Preliminary experiments show that our approach is fast, scalable and can significantly increase the performance (often orders of magnitude) of the SAT engine.

I. INTRODUCTION

Contemporary SAT solvers have matured over the years and come a long way from the DPLL-based chronological backtracking procedures of Davis-Putnam (DP) [4] and Davis-Logemann-Loveland (DLL) [5]. Recent approaches [3] [1] [2] etc., employ sophisticated methods such as constraint propagation and simplification, conflict analysis, learning and non-chronological backtracks [3] [1] [2] to efficiently analyze and prune the search space.

An important aspect of CNF-SAT is to derive an ordering of variables to guide the search. The order in which variables (and correspondingly, constraints) are resolved significantly impacts the performance of SAT search procedures. Most conventional SAT solvers [1] [2] [3] employ variable-activity based branching heuristics to resolve the constraints. Activity of a variable is its frequency of occurrence among the constraints. For a comprehensive review of the effect of activity based branching strategies for SAT solver performance, refer to [6].

Constraint partitioning and minimum-width tree decomposition schemes have been investigated in the context of constraint satisfaction problems [7] [8]. Recently, such approaches have also found application in DPLL-based CNF-SAT search [9] [10] [11][12]. Above approaches analyze and exploit the variable-constraint relationship to derive a variable order for efficient SAT search. However, the computational complexity (exponential) of the proposed algorithms results in large compute times to search for the variable order. As a result, these techniques are impractical for solving large/hard CNF-SAT problems [10].

Research Contributions: This paper proposes hypergraph partitioning based SAT search procedures that attempt to overcome the practical limitations of the above approaches. Instead of relying on the fine-grained (and hence computationally complex) minimum-width tree decomposition procedures, *this research advocates the use of hypergraph partitioning methods [13] to decompose the constraints. Subsequently, this decomposition is analyzed to solve the SAT problem efficiently*. Even though our approach does not directly derive a minimum-width tree decomposition, results show that SAT search can be significantly expedited using our hypergraph partitioning based approach. Moreover, we show that our technique is robust and scalable and can handle a large set of variables and constraints - where contemporary tree decomposition methods take unacceptably long time.

The goals of this research are *two-fold*: 1) to engineer a constraint decomposition technique using hypergraph partitioning; 2) to engineer a constraint resolution method based on this decomposition. First, we propose a novel tri-partitioning of the constraints and derive a procedure to resolve the constraints hierarchically. Sub-problems within the partitions are resolved by propagating the constraints across partitions. We experimentally analyze and comment on the efficacy of our approach. The conclusions derived from these preliminary experiments motivate a *yet another iterative constraint decomposition scheme*. In this approach, the constraints are decomposed in a chain of connected partitions which suggests a variable ordering scheme to guide SAT diagnosis. The variable order derived through the partitioning results in significant increase in performance (often orders of magnitude) of the SAT engine. Our approach is fast and scalable; it is a viable alternative to contemporary minimum-width tree decomposition techniques in the context of deriving a good variable order for SAT search.

II. CONSTRAINT DECOMPOSITION VIA HYPERGRAPH PARTITIONING

First, let us analyze the effect of hypergraph partitioning on the given SAT problem. The given SAT problem in standard DIMACS CNF formulae is converted into a hyper-graph, where variables are represented as hyper-graph edges and clauses are modeled as vertices. A balanced min-cut bi-partitioning is applied. We use the state-of-the-art hypergraph partition tool hMeTiS [13] and port it towards our problem of interest. To search for SAT solutions, we use a modified version of the zCHAFF SAT solver [1]. Figure 1 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 partitions are termed as *LEFT* and *RIGHT* partitions.

Once the original constraint set has been partitioned into two, we need to derive a technique to solve the SAT problem for the respective partitions. Moreover, the solution obtained from individual partitions needs to be reconciled with the other. If the number of cut-set variables is very small, then we can make assignments to these variables and solve the partitions independently. However, in the worst-case it would require an exponential number of assignments to the cut-set variables (backtracks). We have observed from our preliminary experiments that, in most cases, the number of cut-set variables is relatively large. Therefore, such a brute-force technique is inefficient. In order to solve the partitioned SAT problem efficiently, the "partitioning statistics" need to be analyzed further.

During the course of these experiments, we made an interesting observation related to the activity statistics of the cut-set variables. 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. In our experiments, we observed that the variables having higher activity formed the cut-set. We analyzed the reason why the cut-set variables have high activity. This can be *intuitively* explained as follows: High activity variables appear in a larger number of clauses. Balanced bi-partitioning distributes these clauses (containing high activity variables) across the partitions. Therefore, high activity variables form the cut-set, whereas the low activity variables are grouped within respective partitions. It is important to begin the search by making assignments to variables of high activity [6]

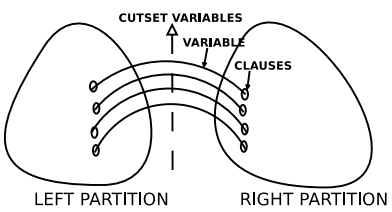
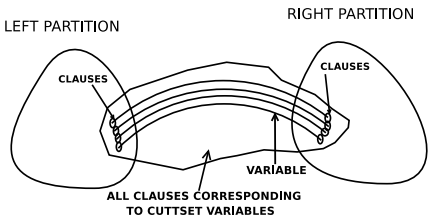


Fig. 1. Balanced bi-partitioning



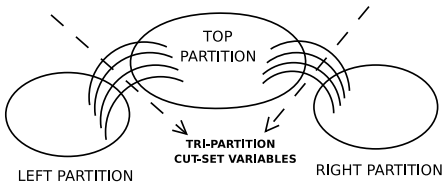


Fig. 2. Tri-partition: Subproblem with high activity variables extracted from the bipartition

[1]. This implies that the cut-set variables should be the first choice for case-splitting. Keeping this in mind, we propose a *tri-partitioning* scheme to solve the partitioned SAT problem.

A. Tri-Partition Derived from Balanced Bi-Partition

In this experiment, a third partition is created from the previously derived *balanced* bipartition as follows. The vertices (clauses) corresponding to the cut-set variables are extracted from both the partitions (LEFT and RIGHT) and are collected together to form a third partition (called TOP partition) as shown in Fig. 2. As a result, the extracted subproblem corresponds to clauses with high activity variables. Moreover, the LEFT and RIGHT partitions have no directly connecting hyperedges. We propose the following method for solving SAT on the above derived tri-partition structure.

1. The TOP PARTITION CNF formulae are first given to the SAT solver.
2. If the solution to this partitioned sub-problem is an *Unsatisfiable* instance, then the original problem is also Unsatisfiable.
3. If a solution to TOP partition is found, then LEFT and RIGHT partitions are both constrained with the assignments to the their respective cut-set variables. Subsequently, the updated LEFT partition is given to the SAT solver.
4. If the solution to LEFT partition is found, the search is transferred to RIGHT partition. If a solution to RIGHT partition is also found, then the original problem is satisfiable.
5. If a solution is not found for the updated constraints in LEFT/RIGHT partition, then the search backtracks to TOP partition and adds a conflict induced clause corresponding to the cut-set variables. For example, if $a = b = 1$ was the assignment to the cut-set variables, which resulted in an UNSAT instance for LEFT and/or RIGHT partition, then a conflict clause $a' + b'$ is added to the TOP partition, and the search is restarted.
6. This procedure of backtracking between partitions (BBP) is repeated until: i) a solution is found; or ii) TOP partition becomes an UNSAT instance, in which case the problem is unsatisfiable.

Some results are presented in the Table, which is shown in Fig. 3. In the table, column "LEFT/RIGHT" corresponds to the balanced bi-partition statistics, i.e., the number of clauses in corresponding partitions. Column "TOP Vars/Clauses" corresponds to the Variables/Clauses in the subsequently extracted TOP partition. The FPGA routing benchmarks are SAT instances, whereas the Urquhart problems are UNSAT instances.

Note that original zCHAFF takes significant amount of search time to find a solution to the FPGA routing problems. On the other hand, using our partitioned approach, we have been able to solve these benchmarks within one second including the time to partition and time to backtrack between partitions. However, the performance of our technique on UNSAT instances is poor when compared to the monolithic SAT technique. Even the small Urq3_1 benchmark cannot be solved in less than 1000 seconds. Why is it that the partitioning scheme does not provide good results for UNSAT problems? We analyzed the results and inferred that solving TOP partition independently always provides a partial solution that cannot be reconciled with LEFT and RIGHT partitions. As a result, our approach requires a large number of backtracks between partitions (BBP) to prove unsatisfiability. Also, we have observed that for the UNSAT instances, BBP increases exponentially with the variables in TOP partition.

Recall that our tri-partition was derived from a balanced bi-partition. A balanced bi-partition resulted in a large number of cut-set variables. Our tri-partitioned approach backtracks on the assignments made to these variables. One way to overcome this problem of exponential BBP is to reduce the number of cut-set variables by unbalancing the partitions. We have performed experiments by unbalancing the partitions[1]. We found that while unbalanced partitioning improves the performance on UNSAT instances; however, it degrades the performance on SAT instances. We analyzed the reason for such a behaviour, which is elaborated below.

When the partitioning is unbalanced, the bi-partition cut-set is generally formed by low-activity variables and the cut-set size also reduces. The proposed tri-partitioned SAT procedure now case splits on these low activity variables first. This results in poor performance for SAT instances. On the other hand, for the UNSAT instances, because of the reduction in cut-set size, the backtracking between partitions also decreases. Hence, the speed-up for UNSAT instances.

Another way to analyze the above issue is that of variable ordering for SAT search. Our approach of partitioning the constraints results in an *order* in which the constraints (and correspondingly, variables) are resolved. Therefore, the above experiments motivated us to ask this question: Can the above partitioning scheme be extended in such a way so as to derive an ordering of variables to guide SAT diagnosis? This question is

[1]The results are omitted due to space limitations.

Fig. 3. SAT solving on Tri-partitioning based decomposition

Bench-mark	Vars/ Clauses	zCHAFF (sec)	Part. Time(s)	LEFT/ RIGHT	TOP Vars/ Clauses	BBP	Total Time (s)
fpga12_9	162 / 684	1591.44	0.262	342 / 342	54/399	1	**0.787**
fpga13_10	195 / 905	>1000	0.289	452 / 453	66/529	1	**1.284**
fpga13_12	234 / 1242	>1000	0.395	621 / 621	79/753	1	**1.329**
Urq3_1	43 / 334	**47.51**	0.178	167 / 167	17/320	32768	1027.1
Urq3_10	37 / 236	**3.85**	0.106	118 / 118	15/230	8192	55.189

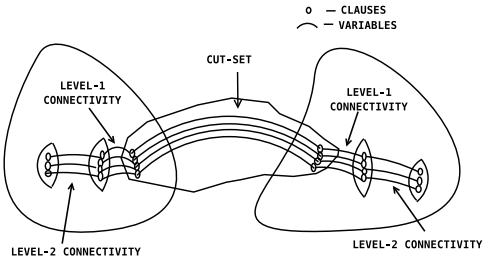


Fig. 4. Analyzing Clause-Var Dependencies

answered in the following section.

III. Tree Decomposition based on Hypergraph partitioning: A variable order for SAT search

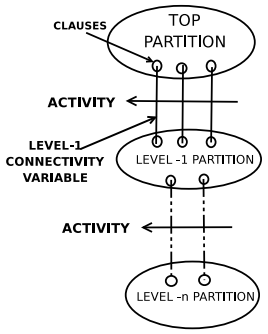


Fig. 5. Fully decomposed SAT problem

The importance of branching on high active variables for SAT is well known [6] [1] [2], and it is also observed in our previous experiments. Moreover, partitioning based constraint resolution highlights the importance of analyzing constraint-variable dependencies. Therefore, we extend our partitioning scheme by iteratively decomposing the constraints *by analyzing both variable activity along with their constraint dependency*. The resulting tree-like decomposition provides a variable order for guiding CNF-SAT search. Our procedure is explained below.

As shown in Fig. 2, 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 (var_ord_list). The SAT tool will branch on these variables first. Note that, the clauses in top-level partition also contain a set of variables, other than those of the cut-set, which correspond to the first level of connectivity among constraints. This is shown in Fig. 4 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 (var_ord_list). Repeating the above procedure, results in a fully decomposed tree as shown in Fig. 5. This var_ord_list provides an order for SAT search.

IV. Experimental Results and Analysis

The above approach has been programmed as an algorithm which is integrated with both hMeTiS [13] and zCHAFF [1]. Experiments were conducted over a wide range of satisfiable, as well as unsatisfiable benchmarks from: (i) Miter circuits (UNSAT instances); (ii) FPGA routing benchmarks (SAT); (iii) Urquhart problems (UNSAT); (iv) Velev's micro-processor verification benchmarks (UNSAT). The results are analyzed below.

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

Bench-mark	Vars/ Clauses	zCHAFF solve time(s)	Amir Part. time(s)	MINCE Part. time(s)	Ours Part. time(s)
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

First, we compare the time required to derive the variable order by our approach with contemporary partitioning based approaches - that of Amir *et al.* [8] and MINCE [12]. Some results for the larger and harder-to-solve CNF-SAT instances are presented in the Table I. As it can be observed 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 contemporary methods; as such Amir's and MINCE approach are too expensive to be applicable for large CAD problems.

Table II demonstrates 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. It is clearly seen from the table that the run times (decomposition time + solve time) of our proposed approach are significantly smaller than that of original zCHAFF SAT solver, even for the larger and more difficult instances. 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.

In order to show that our technique can significantly improve the performance of any DPLL-based SAT engine, we ran the same set of experiments with the MiniSAT solver [14]. The results are presented in the last two columns of Table II.

TABLE II
RUN-TIME COMPARISON OF OUR PROPOSED APPROACH WITH ZCHAFF/MINISAT

		Original zCHAFF			Modified zCHAFF					MiniSAT	
Bench-mark	Vars/ Clauses	Time (sec)	Deci-sions	Implica-tions	Partition Time(s)	Solve Time(s)	Total Time(s)	Deci-sions	Implica-tions	Orig-inal	Modi-fied
fpga12_8	144 / 560	244.42	279,070	5,749,638	0.173	0.41	**0.583**	5,674	95,031	0	0.01
fpga12_9	162 / 684	>1000	—	—	0.199	1.13	**1.329**	14,095	295,693	0	0.02
fpga12_11	180 / 820	>1000	—	—	0.278	3.46	**3.738**	26,521	433,912	0	0.02
fpga12_12	198 / 968	727.44	455,458	7,846,633	0.02	0.288	**0.3**	1,679	16,670	0	0.01
fpga13_10	195 / 905	>1000	—	—	1.14	0.231	**1.371**	12,949	219,829	0.05	0.03
fpga13_12	234 / 1242	>1000	—	—	0.06	0.353	**0.413**	2,250	32,698	0	0
Urq3_1	43 / 334	112.05	1,053,197	12,730,779	0.102	7.81	**7.912**	174,977	1,109,054	12.5	9.91
Urq3_4	36 / 220	**0.07**	6,098	38,021	0.083	0.08	0.163	4,837	45,992	0.17	0.2
Urq3_9	37 / 236	3.37	80,290	1,025,862	0.076	1.68	**1.756**	41,869	351,605	0.54	0.69
Urq3_10	37 / 236	3.37	80,290	1,025,862	0.076	0.93	**1.006**	25,012	173,403	0.53	0.27
c880_opt	770 / 2126	1.13	16,911	812,548	0.362	0.33	**0.392**	11,477	432,012	0.53	0.63
c1355_opt	1006 / 2954	13.62	98,931	8,560,559	0.549	0.46	**1.009**	14,659	631,135	0.37	0.76
c1908_opt	1895 / 5023	**1.67**	20,014	263,4902	0.9	0.82	1.72	14,053	1,508,262	1.53	1.32
c2670_opt	2527 / 6438	**1.48**	45,713	1,921,714	1.15	1.38	2.53	53,150	1,993,412	1.1	1.16
c3540_opt	3431 / 9262	**20.57**	74,325	16,600,030	1.42	22.22	23.64	83,304	18,984,371	40.04	26.5
c5315_opt	4992 / 14151	34.62	136,531	19,829,630	1.25	13.35	**14.6**	128,008	13,129,260	56.65	24.32
c7552_opt	5466 / 15150	105.97	384,776	42,128,278	1.76	39.7	**41.46**	257,514	25,418,457	33.96	48.32
3pipe	2468 / 27533	**1.67**	32,816	1,956,556	4.27	3.27	7.54	54,194	3,402,468	6.24	3.77
4pipe	5237 / 80213	111.2	471,592	71,488,318	13.84	74.27	**88.11**	415,351	53,194,998	139.57	55.03
5pipe	9471 / 195452	167.22	1,773,807	94,373,254	38.40	86.76	**125.16**	875,782	46,788,886	68.29	51.15
3pipe_k	2391 / 27405	**2.52**	48,902	2,796,441	3.526	2.4	5.926	43,815	2,478,836	6.73	2.81
4pipe_k	5095 / 79489	184.2	711,615	106,337,088	12.327	51.62	**63.947**	256,708	38,272,072	186.91	91.8
5pipe_k	9330 / 189109	764.47	1,823,949	417,340,698	39.43	384.57	**424.0**	1,337,479	240,184,009	>1000	691.42
3pipe_q0_k	2476 / 25181	8.6	123,615	5,294,597	4.02	1.51	**5.53**	40,940	2,009,529	4.51	2.11
4pipe_q0_k	5380 / 69072	56.83	394,973	51,775,049	13.133	24.59	**37.723**	227,948	26,294,080	48.04	20.15
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	653.82	121.57

V. CONCLUSION AND STATUS OF THE WORK

This paper has presented constraint partitioning schemes that can be employed for efficient constraint resolution via CNF-SAT. We have demonstrated that hypergraph partitioning based approaches can be successfully employed to solve large and hard CNF-SAT problems. This is particularly important for design validation problems in VLSI-CAD that often have a significantly large number of variables and clauses. Our approach is fast, robust, scalable and it generates a good variable order for SAT search.

Future work: Overcoming the limitations of Our Approach - The generated variable order is dynamically modified by zCHAFF SAT solver according to its VSIDS heuristics. As the conflict clauses are added to the database, the variable activity, as well as clause connectivity, changes. Hence, with the increase in number of backtracks, the dynamically updated variable order (due to VSIDS) might deviate from the one derived statically by our approach. Moreover, the quality of the derived variable order depends on the hypergraph partitioning tools. We are currently working to overcome the above limitations: 1) Instead of relying on hMeTiS, we are currently developing a constraint decomposition heuristic by directly analyzing the topology of hypergraph (constraints); 2) We are also implementing a scheme to dynamically update/re-compute the variable order according to our proposed technique, as and when conflict clauses are added and during search restarts.

REFERENCES

[1] M. Moskewicz, C. Madigan, L. Zhao, and S. Malik, "CHAFF: Engineering and Efficient SAT Solver", *in DAC*, 2001.

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

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

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

[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] E. Amir and S. McIlraith, "Partition-Based Logical Reasoning", *in 7th Intl. Conf. on Prin. of Knowledge Represent. and Reasoning*, 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 ICCAD*, pp. 286–292. IEEE Press, 2001.

[12] F. Aloul, I. Markov, and K. Sakallah, "Mince: A static global variable-ordering for sat and bdd", *in IWLS*, 2001.

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

[14] N. Eén and N. Sörensson, "An Extensible SAT Solver", *in 6th International Conference, SAT*, 2003.