Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability

Luís Baptista and João Marques-Silva

Department of Informatics,
Technical University of Lisbon,
IST/INESC/CEL, Lisbon, Portugal
{lmtb,jpms}@algos.inesc.pt

Abstract. This paper addresses the interaction between randomization, with restart strategies, and learning, an often crucial technique for proving unsatisfiability. We use instances of SAT from the hardware verification domain to provide evidence that randomization can indeed be essential in solving real-world satisfiable instances of SAT. More interestingly, our results indicate that randomized restarts and learning may cooperate in proving both satisfiability and unsatisfiability. Finally, we utilize and expand the idea of algorithm portfolio design to propose an alternative approach for solving hard unsatisfiable instances of SAT.

1 Introduction

Recent work on the Satisfiability Problem (SAT) has provided experimental and theoretical evidence that randomization and restart strategies can be quite effective at solving hard satisfiable instances of SAT [4]. Indeed, backtrack search algorithms, randomized and run with restarts, were shown to perform significantly better on specific problem instances. Recent work has also demonstrated the usefulness of learning in solving hard instances of SAT [2, 6, 8]. Learning, in the form of clause (*nogood*) recording, is the underlying mechanism by which non-chronological backtracking, relevance-based learning, and other search pruning techniques, can be implemented.

In this paper we propose to conduct a preliminary study of the interaction between randomization and learning in solving real-world hard satisfiable instances of SAT. Moreover, we propose a new problem solving strategy for solving hard unsatisfiable instances of SAT. Throughout the paper we focus on real-world instances of SAT from the hardware verification domain, namely superscalar processor verification [7][1]. These instances can either be satisfiable or unsatisfiable and are in general extremely hard for state of the art SAT solvers.

2 Randomization and Learning

A complete backtrack search SAT algorithm is *randomized* by introducing a fixed or variable amount of randomness in the branching heuristic [4]. The

[1] The superscalar processor verification instances can be obtained from the URL http://www.ece.cmu.edu/∼mvelev.

Table 1. Results for Grasp with Restarts and without Learning and for SATZ

Instance	Grp 0/0C 100/0		Grp 0/0C 500/250		Satz 20000		Satz 1000		Satz 100	
	Time	X	Time	X	Time	X	Time	X	Time	X
2dlx_cc_bug1	790.0	1	2464.7	8	3000	10	3000	10	3000	10
2dlx_cc_bug105	7.3	0	120.2	0	3000	10	3000	10	3000	10
2dlx_cc_bug11	172.1	0	1702.4	4	3000	10	3000	10	3000	10
2dlx_cc_bug38	166.4	0	2121.8	6	3000	10	3000	10	3000	10
2dlx_cc_bug54	1322.2	3	2859.9	9	1208	2	359.0	1	142.2	0
2dlx_cc_bug80	397.5	0	1665.4	5	3000	10	3000	10	3000	10
dlx2_cc_a_bug5	1652.6	3	3000.0	10	3000	10	3000	10	3000	10
dlx2_cc_a_bug59	278.3	0	2099.0	6	657	0	79.4	0	41.4	0
dlx2_cc_bug02	1140.1	2	2768.2	8	3000	10	3000	10	3000	10
dlx2_cc_bug08	426.3	0	2852.8	9	3000	10	3000	10	3000	10

amount of randomness may affect the value of the selected variable, which variable is selected from the set of variables with the highest heuristic metric, or even which variable is selected from a set of variables within $x\%$ of the highest value of the heuristic metric. Morever, a *restart strategy* consists of defining a *cutoff* value in the number of backtracks, and repeatedly running a randomized complete SAT algorithm, each time limiting the maximum number of backtracks to the imposed cutoff value.

If randomized restarts are used with a fixed cutoff value, then the resulting algorithm is *not* complete. Even though the resulting algorithm has a non-zero probability of solving every satisfiable instance, it may not be able to prove instances unsatisfable. One simple solution to this limitation, that allows solving unsatisfiable instances, is to implement a policy for increasing the cutoff value. For example, after each restart the backtrack cutoff value can be increased by a constant amount. The resulting algorithm is complete, and thus able to prove unsatisfiability.

Clause (*nogood*) recording (i.e. learning) techniques are currently the foundation upon which modern backtrack search algorithms [2, 6, 8] build to implement different search pruning techniques, including non-chronological backtracking, relevance-based learning, among others. If an algorithm implements branching randomization and a restart strategy, then each time the cutoff limit on the number of backtracks is reached, new clauses are expected to have been identified. These clauses may either be discarded or kept for subsequent restarts of the algorithm. Clearly, one can limit the size of the clauses that are to be kept in between restarts. Below, we study how useful the different aspects of learning are when randomization with a restart strategy is used.

Next we compare GRASP [6] and SATZ [5] with randomization and restarts (results without restarts that involve these and other algorithms, as well as additional results with restarts, are analyzed in [1])[2]. For each instance and for each algorithm the number of runs was limited to 10. A significantly larger number would have required excessive run times.

[2] The results were obtained on a P-II 400 MHz Linux machine with 256 MByte of physical memory. The CPU time limit was set to 3,000 seconds. Column `Time` denotes the CPU time and column `X` denotes the number of times, out of the total number of runs, the algorithm was *not* able to solve the given instance.

Table 2. Results with Restarts and Learning (cutoff/increment = 500/250)

Instance	Grasp 0/0		Grasp 0/10		Grasp 0/20		Grasp 10/20		Grasp 20/20	
	Time	X	Time	X	Time	X	Time	X	Time	X
2dlx_cc_bug1	57.0	0	67.0	0	105.9	0	66.3	0	196.2	0
2dlx_cc_bug105	19.6	0	22.2	0	25.5	0	33.7	0	51.4	0
2dlx_cc_bug11	240.4	0	338.8	0	399.1	0	168.5	0	226.9	0
2dlx_cc_bug38	44.4	0	54.4	0	50.3	0	73.6	0	100.4	0
2dlx_cc_bug54	252.6	0	228.1	0	198.1	0	166.8	0	143.3	0
2dlx_cc_bug80	127.3	0	54.7	0	50.5	0	41.5	0	59.8	0
dlx2_cc_a_bug5	133.9	0	121.9	0	206.1	0	151.0	0	204.4	0
dlx2_cc_a_bug59	33.2	0	14.2	0	24.3	0	17.8	0	17.3	0
dlx2_cc_bug02	147.6	0	49.9	0	47.0	0	42.9	0	91.9	0
dlx2_cc_bug08	48.8	0	27.0	0	19.8	0	25.7	0	29.8	0

Table 1 contains the results of running GRASP with chronological backtracking enabled (C), and with no clauses being recorded, and the results for the randomized version of SATZ [4, 5]. For SATZ the cutoff values used were 20000, 1000 and 100 backtracks. For GRASP, the two inital cutoff and increment values considered were 100/0 and 500/250. As can be concluded, SATZ is only able to solve two instances for any of the cutoff values considered, and for these two instances, the CPU times decrease with smaller cutoff values. In addition, SATZ exceeds the allowed CPU time for all the other instances and for the different cutoff values considered. The results for GRASP depend on the cutoff initial and increment values. The utilization of the combination 100/0 is clearly better for these problem instances. Nevertheless, the algorithm quits in a few cases and for some runs.

Table 2 contains the results of running GRASP with non-chronological backtracking enabled, using randomization and restarts, and with different clause recording arrangements. Each column is identified by two values ws/g, denoting the largest clause size that is kept in between restarts (ws), and the largest clause size that GRASP records during the search (g). For this experiment the initial cutoff value was set to 500, and the increment set to 250. The conclusions are clear. Branching randomization with a restart strategy allows solving *all* problem instances for *all* runs, provided learning is enabled. Moreover, and in most of the examples, recording clauses both during the search and in between restarts, can contribute to reducing the CPU times.

3 Algorithm Portfolio Design

Recent work on algorithm portfolio design [3] has shown that a portfolio approach for solving hard instances of SAT can lead to significant performance gains. Basically, a set of algorithms is selected which is then used for solving each problem instance on different processors, or interleaving the execution of several algorithms in one or more processors. In this section we explore this idea with the objective of solving hard *unsatisfiable* instances of SAT. As before, the problem instances studied were obtained from hardware verification problems [7].

Our portfolio approach is somewhat different than what was proposed in [3], and targets proving unsatisfiability for hard instances of SAT. Instead of having fundamentally different algorithms, or several copies of the same algorithm running on different processors, our approach is to utilize randomization with a restart strategy, and each time the search is restart, a *different* algorithm is selected from a set of k different algorithms $\{A_1, \ldots, A_k\}$. Each algorithm $\{A_i\}$ has a given probability p_i of being selected.

The key aspects that characterize our portfolio approach are that the restart strategy used is to iteratively increase the cutoff limit, thus guaranteeing completeness of the algorithm, and that learning between restarts is used, thus reusing information from previously searched portions of the search tree to avoid subsequently searching equivalent portions.

In our current solution, instead of using significantly different search algorithms, we utilize instead significantly different configurations of the same algorithm, GRASP. Different configurations of GRASP basically allow different branching heuristics, different amounts of randomization in the branching heuristics, different learning bounds during the search, and different learning bounds in between restarts.

For the results presented below, the portfolio of configurations considered was the following:

- Four different configurations with similar probabilities are used.
- The limit on recorded clauses during search ranges from 30 to 40.
- Relevance-based learning ranges from 4 to 5.
- Recorded clauses in between restarts range from 10 to 15.
- The amount of randomization is fixed for all configurations.
- Three well-known and widely used branching heuristics are used (see [1]).

The experimental results for the unsatisfiable hardware verification instances [7] are shown in Table 3. For this experiment, we evaluated two portfolio organizations, i.e. `pf1` and `pf2`, that differ in the initial cutoff and increment values. For `pf1` the values were set to 500/250, and for `pf2` the values were set to 100/50.

The results for the default GRASP algorithm results are shown in column `Grasp`, the results for the restart strategy in column `Grasp (rst)`, and the results for the portfolio approach in columns `Grasp (pf1)` and `Grasp (pf2)`. In addition, the results for rel_sat, SATZ and randomized SATZ (`Satz (rst)`) are also shown. In all cases the CPU time limit was 10,000 seconds, with the exception of the default GRASP algorithm for which 100,000 seconds were allowed. Finally, and as in the previous section, the total number of runs was 10 for the algorithms using randomization with restarts.

As can be concluded, the portfolio approach, built on top of branching randomization with restarts, allows solving *all* instances with much smaller run times (for the harder instances more than 1 order magnitude faster). Of the two organizations evaluated, the best results were obtained with the initial cutoff and increment values set to 500/250, since on average this choice of values allows solving the harder instances faster. We should note that, to our best knowledge, no other SAT algorithm is capable of solving the harder instances, `dlx2_cc` and `dlx2_cl`, thus suggesting that an approach based on a portfolio of configurations may be crucial for proving unsatisfiability for some classes of instances.

4 Conclusions

This paper studies the interaction between randomization and learning in backtrack search SAT algorithms when solving real-world hard instances of SAT.

Table 3. Results on Unsatisfiable Instances

File	Grasp		Grasp (rst)		Grasp (pf1)		Grasp (pf2)		Relsat		Satz		Satz (rst)	
	Time	X	Time	X	Time	X	Time	X	Time	X	Time	X	Time	X
dlx1_c	1.9	0	2.2	0	3.6	0	3.7	0	7.0	0	7918.7	0	10000	10
dlx2_aa	1.8	0	2.7	0	17.5	0	9.5	0	23.7	0	10000	1	10000	10
dlx2_ca	2686.7	0	3006.1	0	631.3	0	758.2	0	10000	1	10000	1	10000	10
dlx2_cc	100000	1	10000	10	2032.1	0	2401.6	0	10000	1	10000	1	10000	10
dlx2_cl	100000	1	10000	10	1076.4	0	1077.9	0	10000	1	10000	1	10000	10
dlx2_cs	9598.1	0	8970.4	6	987.0	0	1263.3	0	10000	1	10000	1	10000	10
dlx2_la	6259.2	0	9617.4	7	307.7	0	382.0	0	10000	1	10000	1	10000	10
dlx2_sa	12.0	0	7.6	0	34.2	0	29.0	0	295.6	0	10000	1	10000	10

Preliminary results indicate that both randomization and learning (in the form of non-chronological backtracking ability and recorded clauses) can be essential for solving the satisfiable problem instances studied. Moreover, and for unsatisfiable instances, we have provided empirical evidence that randomization and learning, when utilizing a portfolio of algorithm configurations, may solve problem instances that, to our best knowledge, no known SAT algorithm is otherwise able to solve. Finally, the experimental results obtained indicate that significantly different organizations (in terms of the initial cutoff and increment values) may be required for proving either satisfiability or unsatisfiability. Future work will necessarily address developing unified organizations for proving both satisfiability and unsatisfiability in classes of problem instances.

References

1. L. Baptista and J. P. Marques-Silva. The interplay of randomization and learning on real-world instances of satisfiability. In *AAAI Workshop on Leveraging Probability and Uncertatinty in Computation*, July 2000.
2. R. Bayardo Jr. and R. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the National Conference on Artificial Intelligence*, 1997.
3. C. P. Gomes and B. Selman. Algorithm portfolio design: Theory vs. practice. In *Proceedings of the Thirteenth Conference On Uncertainty in Artificial Intelligence*, 1997.
4. C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the National Conference on Artificial Intelligence*, July 1998.
5. C. M. Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proceedings of International Conference on Principles and Practice of Constraint Programming*, 1997.
6. J. P. Marques-Silva and K. A. Sakallah. GRASP-A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.
7. M. N. Velev and R. E. Bryant. Superscalar processor verification using efficient reductions from the logic of equality with uninterpreted functions to propositional logic. In *Proceedings of Correct Hardware Design and Verification Methods*, LNCS 1703, pages 37–53, September 1999.
8. H. Zhang. SATO: An efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction*, pages 272–275, July 1997.