Complete Search Restart Strategies for Satisfiability

L. Baptista and **I. Lynce** and **J. Marques-Silva** [1]
{lmtb,ines,jpms}@sat.inesc.pt
Technical University of Lisbon, INESC/CEL
Lisbon, Portugal

Abstract

Search restarts is a well-known strategy for coping with hard real-world satisfiable (and often unsatisfiable) instances of Propositional Satisfiability (SAT), that is already being used by different state-of-the-art SAT solvers. Despite being extremely effective for solving real-world problem instances, the original search restart strategy yields an incomplete algorithm, that for unsatisfiable instances is unable to guarantee that unsatisfiability can be established. This paper describes different techniques for guaranteeing the completeness of SAT algorithms that implement search restart strategies, and proposes implementation optimizations to some of the proposed techniques. Experimental evidence indicates that the proposed techniques are effective for a wide range of real-world problem instances.

1 Introduction

Backtrack search algorithms for Propositional Satisfiability (SAT) have seen significant improvements in recent years [Bayardo Jr. and Schrag, 1997; Gomes *et al.*, 1998; Moskewicz *et al.*, 2001; Marques-Silva and Sakallah, 1996; Prestwich, 2000; Zhang, 1997]. These improvements result from new search pruning techniques as well as new strategies for how to organize the search. Effective search pruning techniques include, among others, clause recording and non-chronological backtracking [Bayardo Jr. and Schrag, 1997; Marques-Silva and Sakallah, 1996; Moskewicz *et al.*, 2001], whereas recent effective strategies include restarts [Gomes *et al.*, 1998] and also randomized backtracking [Prestwich, 2000].

Intrinsic to several of these improvements is randomization. Randomization has found application in different SAT algorithms, that include local search and backtrack search algorithms [McAllester *et al.*, 1997; Bayardo Jr. and Schrag, 1997]. In backtrack search, current state-of-the-art SAT solvers extensively resort to randomization, most often for selecting variable assignments but (and as a result) also within search restart strategies. Moreover, the recent work by S. Prestwich [Prestwich, 2000] (though preceded by the work of others [Ginsberg and McAllester, 1994]) has motivated the utilization of randomly picked backtrack points in SAT algorithms.

The paper has three main objectives. First, to describe how to eliminate search redundancy when search restarts are utilized, and simultaneously guaranteeing that the resulting algorithm is complete. We should note that completeness comes at a cost, since the growth of the problem instance size can be exponential in the worst-case. Second, to propose optimizations that tackle the potential growth problem of the instance being solved. Third, and finally, to provide empirical evidence that restart without repeated search paths can lead to significant savings in the amount of search effort.

Search restarts have been proposed and shown effective for real-world instances of SAT by the work of Gomes, Selman and Kautz [Gomes *et al.*, 1998]. The proposed algorithm is not complete, since the restart cutoff point is kept constant. In [Richards and Richards, 1997] and [Baptista and Marques-Silva, 2000], search restarts were used jointly with learning, and both algorithms guarantee completeness. Nevertheless, whereas the first algorithm is complete since *all* recorded clauses are kept, the second algorithm is complete since the backtrack cutoff value increases after each restart. More recently, in [Moskewicz *et al.*, 2001] search restarts were utilized and completeness was ensured by increasing the restart cutoff point. Moreover, in [Moskewicz *et al.*, 2001] it is also suggested keeping all recorded clauses or, alternatively, increasing the relevance-based learning parameter [Bayardo Jr. and Schrag, 1997] with each restart. Both techniques ensure completeness.

The remainder of this paper is organized as follows. Section 2 presents definitions used throughout the paper. Afterwards, Section 3 briefly surveys SAT algorithms and the utilization of randomization in SAT. Next, in Section 4 we describe the improvements to

[1] Corresponding Author.
Address: R. Alves Redol, 9, Sala 522
1000-029 Lisboa, Portugal
Phone: 351-21-310-0387 Fax: 351-21-314-5843
Web: http://sat.inesc.pt/~jpms

search restarts proposed in this work. Preliminary experimental results are presented and analyzed in Section 5, and the paper concludes in Section 6.

2 Definitions

This section introduces the notational framework used throughout the paper. Propositional variables are denoted $x_1, \ldots, x_n$, and can be assigned truth values 0 (or F) or 1 (or T). The truth value assigned to a variable x is denoted by $\nu(x)$. A literal l is either a variable x_i or its negation $\neg x_i$. A clause ω is a disjunction of literals and a CNF formula φ is a conjunction of clauses. A clause is said to be *satisfied* if at least one of its literals assumes value 1, *unsatisfied* if all of its literals assume value 0, *unit* if all but one literal assume value 0, and *unresolved* otherwise. Literals with no assigned truth value are said to be *free literals*. A formula is said to be satisfied is all its clauses are satisfied, and is unsatisfied if at least one clause is unsatisfied. The SAT problem is to decide whether there exists a truth assignment to the variables such that the formula becomes satisfied.

It will often be simpler to refer to clauses as sets of literals, and to the CNF formula as a set of clauses. Hence, the notation $l \in \omega$ indicates that a literal l is one of the literals of clause ω, whereas the notation $\omega \in \varphi$ indicates that clause ω is one of the clauses of CNF formula φ.

In the following sections we shall address backtrack search algorithms for SAT. Most if not all backtrack search SAT algorithms apply extensively the *unit clause rule* [Davis and Putnam, 1960]. If a clause is unit, then the sole free literal must be assigned value 1 for the formula to be satisfiable. The iterated application of the unit clause rule is often referred to as Boolean Constraint Propagation (BCP) [Zabih and McAllester, 1988]. For implementing some of the techniques common to some of the most competitive backtrack search algorithms for SAT, it is necessary to properly *explain* the truth assignments to the propositional variables that are implied by the clauses of the CNF formula. For example, let $x = v_x$ be a truth assignment implied by applying the unit clause rule to a unit clause clause ω. Then the explanation for this assignment is the set of assignments associated with the remaining literals of ω, which are assigned value 0.

Let $\omega = (x_1 \vee \neg x_2 \vee x_3)$ be a clause of a CNF formula φ, and assume the truth assignments $\{x_1 = 0, x_3 = 0\}$. Then, for the clause to be satisfied we must necessarily have $x_2 = 0$. We say that the implied assignment $x_2 = 0$ has the explanation $\{x_1 = 0, x_3 = 0\}$. A more formal description of explanations for implied variable assignments, as well as a description of mechanisms for their identification, can be found for example in [Marques-Silva and Sakallah, 1996].

3 SAT Algorithms

Over the years a large number of algorithms have been proposed for SAT, from the original Davis-Putnam procedure [Davis and Putnam, 1960], to recent backtrack search algorithms [Bayardo Jr. and Schrag, 1997; Marques-Silva and Sakallah, 1996; Zhang, 1997; Moskewicz *et al.*, 2001], to local search algorithms [Selman and Kautz, 1993], among many others.

SAT algorithms can be characterized as *complete* and as *incomplete*. Complete algorithms can establish unsatisfiability if given enough CPU time; incomplete algorithms cannot. In a search context complete algorithms are often referred to as *systematic*, whereas incomplete algorithms are referred to as *non-systematic*.

Among the different algorithms, we believe backtrack search to be the most robust approach for solving hard, structured, real-world instances of SAT. This belief has been amply supported by extensive experimental evidence obtained in recent years [Baptista and Marques-Silva, 2000; Marques-Silva and Sakallah, 1996; Moskewicz *et al.*, 2001].

3.1 Backtrack Search SAT Algorithms

The vast majority of backtrack search SAT algorithms build upon the original backtrack search algorithm of Davis, Logemann and Loveland [Davis *et al.*, 1962]. Recent state-of-the-art backtrack search SAT solvers [Bayardo Jr. and Schrag, 1997; Marques-Silva and Sakallah, 1996; Zhang, 1997; Moskewicz *et al.*, 2001] utilize sophisticated variable selection heuristics, fast Boolean Constraint Propagation procedures, and incorporate techniques for diagnosing conflicting conditions, thus being able to backtrack non-chronologically and record clauses that explain and prevent identified conflicting conditions.

3.2 Applying Randomization

The utilization of different forms of randomization in SAT algorithms has seen increasing acceptance in recent years.

Randomization is essential in many local search algorithms [Selman and Kautz, 1993]. Most local search algorithms repeatedly restart the (local) search by randomly generating complete assignments. Moreover, randomization can also be used for deciding among different (local) search strategies [McAllester *et al.*, 1997].

Randomization has also successfully been included in variable selection heuristics of backtrack search algorithms [Bayardo Jr. and Schrag, 1997]. Variable selection heuristics, by being greedy in nature, are unlikely but unavoidably bound to select the wrong variable at the wrong time for the wrong instance. The utilization of randomization helps reducing the probability of seeing this happening.

Although intimately related with randomizing variable selection heuristics, randomization is also a key aspect of restart strategies [Gomes *et al.*, 1998]. The utilization of randomization results in different subtrees being searched each time the search algorithm is restarted.

Finally, randomization has also been used in the backtrack step of incomplete backtrack search algorithms. In

CLS [Prestwich, 2000], the backtracking variable is randomly picked each time a conflict is identified. By not always backtracking to the most recent untoggled decision variable, the CLS algorithm is able to often avoid the characteristic trashing of backtrack search algorithms. We should note, however, that the CLS algorithm is not complete and so cannot establish unsatisfiability.

Current state-of-the-art SAT solvers already incorporate some of the above forms of randomization [Baptista and Marques-Silva, 2000; Moskewicz *et al.*, 2001]. In these SAT solvers variable selection heuristics are randomized and restart strategies are utilized.

4 Complete Restart Strategies

Despite being an effective search technique, restart strategies yield incomplete algorithms. This was addressed in [Baptista and Marques-Silva, 2000], where the backtrack cutoff value is increased after each search restart. The resulting procedure resembles *iterative deepening* and is complete. This approach has recently been incorporated into a highly optimized SAT solver [Moskewicz *et al.*, 2001]. However, as mentioned in Section 1 there are other approaches that guarantee completeness. For example, in [Moskewicz *et al.*, 2001] it is suggested increasing the relevance-based learning threshold with each restart to guarantee completeness. Alternatively, another related approach that also guarantees completeness is to increase the size of recorded clauses that are actually kept in between restarts. Hence, the size of clauses that are deemed large [Marques-Silva and Sakallah, 1999] and are eventually deleted is increased after each restart.

However, all these approaches still exhibit a key drawback: paths in the search tree can still be visited more than once. Observe that, even when clause recording is used, as in [Marques-Silva and Sakallah, 1999; Moskewicz *et al.*, 2001], clauses are eventually deleted and so search paths may be re-visited.

Interestingly, if *all* recorded clauses are kept, as originally proposed by [Richards and Richards, 1997], then no conflict can be repeated during the search, and so no search paths can be repeated. Hence, as long as the search algorithm keeps *all* recorded clauses, no search paths are ever repeated and so even a constant-cutoff restart strategy is guaranteed to be complete.

The main drawback of keeping *all* recorded clauses is that the growth of the CNF formula is linear in the number of explored nodes, and so exponential in the worst-case. Nevertheless, as we shall see in Section 5, experimental evidence justifies keeping all recorded clauses. Moreover, there are several effective techniques to tackle the potential exponential growth of the CNF formula.

The first technique to tackle CNF formula growth is to opportunistically apply subsumption to recorded clauses. This technique is guaranteed to effectively reduce the number of clauses that are kept in between restarts. Nevertheless, we can do better.

Each time a search restart is to be taken, after a conflict is identified, the algorithm defines a path in the search tree. This path consists of all current decision assignments, and all the toggled decision assignments, which have been explained by identified conflicts[2]. Thus, the search path characterizes the sub-tree T_S that has just been searched by the algorithm. Clearly, the explanation of the current conflict, as well as the explanations of all of the failure-driven assertions in the search path, provide a *sufficient* explanation of why sub-tree T_S, that has just been searched, does not contain a solution to the problem instance. Hence, if we *just* record these explanations as new clauses, and delete *all* other clauses that may have been recorded while searching T_S, then we are guaranteed not to search again nodes that were searched in T_S. We refer to this technique as keeping the *search signature* of each sub-tree T_S in between restarts.

The previous technique reduces the number of subsumption operations that have to be applied during the search. If subsumption is not applied, then the proposed technique effectively reduces the total number of recorded clauses, that are effectively kept in between restarts.

5 Experimental Results

This section presents and analyzes preliminary experimental results of the techniques proposed in this paper. A new SAT solver, Quest0.5, is used. Quest0.5 is is built on top of the GRASP SAT solver [Marques-Silva and Sakallah, 1996] and incorporates restarts. For all the experimental results presented in this section a PIII @866MHz Linux machine with 512MByte of RAM was used. The CPU time limit for each instance was set to 3600 seconds. Since randomization was used, the number of runs was set to 10 (due to the large number of problems instances being solved). Moreover, the results shown correspond to the average values for the 10 runs.

1. The first configuration, `config1`, utilizes the script described in [Baptista and Marques-Silva, 2000]. The initial cutoff value is 100 backtracks, and the increment after each restart is 50 backtracks.
2. The second configuration, `config2`, reimplements the same approach as `config1`, but within Quest0.5. In this case the initial cutoff value is also 100 backtracks, and the increment after each restart is also 50 backtracks.
3. The third configuration, `config3`, keeps *all* recorded clauses. The cutoff value is 100, but is kept fixed, since completeness is guaranteed.
4. The fourth configuration, `config4`, is similar to `config3`, but the clauses that are kept are solely the ones that define the searched sub-tree when the search is restarted, only the *search signatures* are identified and kept.

[2] Toggled decision assignments are explained by recorded clauses, that are referred to as failure-driven assertions [Marques-Silva and Sakallah, 1996].

	sss1.0 (40)	sss1.0a (9)	sss2.0 (100)
config1	267	273	1241
config2	532	286	1153
config3	194	126	598
config4	237	71	475

Table 1: CPU Times for different configurations

	sss1.0 (40)	sss1.0a (9)	sss2.0 (100)
config1	111362	74439	430918
config2	141410	63064	356141
config3	64423	29474	203912
config4	68889	23571	161882

Table 2: Decision nodes for different configurations

The main objective of the techniques proposed in this paper is to be effective in solving hard real-world problem instances. Recent examples of such instances are the superscalar processor verification problem instances developed by M. Velev [Velev and Bryant, 1999]. We consider three sets of instances, the first (sss1.0a) with 9 satisfiable instances, the second (sss1.0.sat) with 40 satisfiable instances, and the third (sss2.0) with 100 satisfiable instances.

The results are shown in Tables 1, 2 and 3. Table 1 includes the CPU times for the four configurations, and for the three sets of problem instances. Table 2 includes the number of decision nodes also for the four configurations, and also for the three sets of problem instances. Finally, Table 3 indicates for how many problem instances in each class are the results improved for each configuration and respect to the previous configuration.

The general trend of the results shown is that the elimination of repeated search paths in restarts helps reducing the total number of decision nodes. The improvements in some cases can be significant. We should note that for `config1` the results can be somewhat misleading. Since an external script is used to implement restarts, some of the overhead is not accounted for, and so for `config1` the CPU times shown are lower than in reality.

6 Conclusions

In this paper we introduce new techniques for guaranteeing the completeness of SAT algorithms that incorporate search restarts, and address implementation optimizations of the proposed techniques. Preliminary experimental results clearly indicate that significant savings in search effort can be obtained when search restarts without repeated paths are used.

Future research work involves further evaluation the relationship between search restarts and learning.

	sss1.0 (40)	sss1.0a (9)	sss2.0 (100)
#best2	16	6	44
#best3	17	7	70
#best4	30	6	69

Table 3: Improvements from previous configuration

References

[Baptista and Marques-Silva, 2000] L. Baptista and J. Marques-Silva. Using randomization and learning to solve hard real-world instances of satisfiability. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming*, 2000.

[Bayardo Jr. and Schrag, 1997] 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*, pages 203–208, 1997.

[Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

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

[Ginsberg and McAllester, 1994] M. Ginsberg and D. McAllester. GSAT and dynamic backtracking. In *Proceedings of the International Conference on Principles of Knowledge and Reasoning*, pages 226–237, 1994.

[Gomes *et al.*, 1998] C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the National Conference on Artificial Intelligence*, July 1998.

[Marques-Silva and Sakallah, 1996] J. P. Marques-Silva and K.Ã. Sakallah. GRASP: A new search algorithm for satisfiability. In *Proceedings of the ACM/IEEE International Conference on Computer-Aided Design*, pages 220–227, November 1996.

[Marques-Silva and Sakallah, 1999] J. P. Marques-Silva and K.Ã. Sakallah. GRASP-A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999.

[McAllester *et al.*, 1997] D. McAllester, B. Selman, and H. Kautz. Evidence of invariants in local search. In *Proceedings of the National Conference on Artificial Intelligence*, pages 321–326, 1997.

[Moskewicz *et al.*, 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the Design Automation Conference*, 2001.

[Prestwich, 2000] S. Prestwich. A hybrid search architecture applied to hard random 3-sat and low-autocorrelation binary sequences. In *Proceedings of* *the International Conference on Principles and Practice of Constraint Programming*, pages 337–352, 2000.

[Richards and Richards, 1997] E. T. Richards and B. Richards. Restart-repair and learning: An empirical study of single solution 3-sat problems. In *CP Workshop on on the Theory and Practice of Dynamic Constraint Satisfaction*, 1997.

[Selman and Kautz, 1993] B. Selman and H. Kautz. Domain-independent extensions to GSAT: Solving large structured satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 290–295, 1993.

[Velev and Bryant, 1999] 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.

[Zabih and McAllester, 1988] R. Zabih and D. A. McAllester. A rearrangement search strategy for determining propositional satisfiability. In *Proceedings of the National Conference on Artificial Intelligence*, pages 155–160, 1988.

[Zhang, 1997] H. Zhang. SATO: An efficient propositional prover. In *Proceedings of the International Conference on Automated Deduction*, pages 272–275, July 1997.