Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 155 (2007) 1604–1612
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

# Random backtracking in backtrack search algorithms for satisfiability

I. Lynce, J. Marques-Silva

*Technical University of Lisbon, IST/INESC-ID/CEL, Rua Alves Redol, 9, 1000-029 Lisboa, Portugal*

Received 15 March 2001; received in revised form 25 August 2003; accepted 19 October 2005
Available online 8 December 2006

**Abstract**

This paper proposes the utilization of randomized backtracking within complete backtrack search algorithms for propositional satisfiability (SAT). In recent years, randomization has become pervasive in SAT algorithms. Incomplete algorithms for SAT, for example the ones based on local search, often resort to randomization. Complete algorithms also resort to randomization. These include state-of-the-art backtrack search SAT algorithms that often randomize variable selection heuristics. Moreover, it is plain that the introduction of randomization in other components of backtrack search SAT algorithms can potentially yield new competitive search strategies. As a result, we propose a stochastic backtrack search algorithm for SAT, that randomizes both the variable selection and the backtrack steps of the algorithm. In addition, we relate randomized backtracking with a more general form of backtracking, referred to as unrestricted backtracking. Finally, experimental results for different organizations of randomized backtracking are described and compared, providing empirical evidence that the new search algorithm for SAT is a very competitive approach for solving hard real-world instances.


*Keywords:* Propositional satisfiability; Backtrack search algorithms; Randomization

## 1. Introduction

Propositional satisfiability (SAT) is a well-known NP-complete problem, with extensive applications in Artificial Intelligence, Electronic Design Automation, and many other fields of Computer Science and Engineering.

In recent years, several competitive solution strategies for SAT have been proposed and thoroughly investigated [9,13,14]. Advanced techniques applied to backtrack search algorithms for SAT have achieved remarkable improvements [9,3,10,13–15,21], having been shown to be crucial for solving large instances of SAT derived from real-world applications. Current state-of-the-art SAT solvers incorporate advanced pruning techniques as well as new strategies on how to organize the search. Effective search pruning techniques include, among others, clause recording and non-chronological backtracking [3,9,13,14], whereas recent effective strategies include restarts [10] and (more recently) randomized backtracking [15].

Intrinsic to several of these improvements is randomization. Randomization has found application in different SAT algorithms, including local search [17] and backtrack search algorithms [3]. In backtrack search, most sophisticated

*E-mail addresses:* ines@sat.inesc.pt (I. Lynce), jpms@sat.inesc.pt (J. Marques-Silva).
*URLS:* http://sat.inesc.pt/~ines (I. Lynce), http://sat.inesc.pt/~jpms (J. Marques-Silva).

0166-218X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2005.10.021

SAT solvers extensively resort to randomization, most often for selecting variable assignments but (and as a result) also within search restart strategies [10]. Moreover, recent work by Prestwich [15] (inspired by the previous work of others [8,16,19]) has motivated the utilization of randomly picked backtrack points in incomplete SAT algorithms.

### *1.1. Objectives*

This paper has three main objectives. First, to motivate the generalized utilization of randomization in a complete search strategy, and propose a stochastic backtrack search algorithm for SAT, that involves randomizing both the variable selection heuristic and the backtrack step of the algorithm. Second, to relate randomized backtracking with unrestricted backtracking (UB) [11], a generic framework that captures the organization of different state-of-the-art backtrack search strategies. Third, and finally, to provide empirical evidence that specific formulations of stochastic backtrack search can lead to very significant savings in the amount of search and time effort when solving hard real-world instances of SAT.

### *1.2. Organization of the paper*

The remainder of this paper is organized as follows. Section 2 presents definitions used throughout the paper. Section 3 briefly surveys SAT algorithms and the utilization of randomization in SAT. Afterwards, Section 4 introduces a stochastic backtrack search SAT algorithm. The next section introduces UB. Experimental results are presented and analyzed in Section 6. Finally, we describe related work in Section 7, and conclude in Section 8.

## 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 $v(x)$. A literal $l$ is either a variable $x_i$ or its negation $\neg x_i$. A clause $\omega$ is a disjunction of literals and a CNF formula $\varphi$ 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 if all its clauses are satisfied, and is unsatisfied if at least one clause is unsatisfied. The SAT problem consists of deciding 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 $\omega$, whereas the notation $\omega \in \varphi$ indicates that clause $\omega$ is one of the clauses of the CNF formula $\varphi$.

In the following sections we shall address backtrack search algorithms for SAT. A backtrack search algorithm is implemented by a *search process* that implicitly enumerates the space of $2^n$ possible binary assignments to the $n$ problem variables. Each different truth assignment defines a *search path* within the search space. A *decision level* is associated with each variable selection and assignment. The first variable selection corresponds to decision level 1, and the decision level is incremented by 1 for each new decision assignment.[1] In addition, and for each decision level, the *unit clause rule* [5] is applied. If a clause is unit, then the sole free literal must be assigned value 1 for the formula to be satisfied. In this case, the value of the literal and of the associated variable are said to be *implied*. The iterated application of the unit clause rule is often referred to as Boolean constraint propagation (BCP) [20].

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 $\omega$. Then the explanation for this assignment is the set of assignments associated with the remaining literals of $\omega$, which are assigned value 0. As an example, let $\omega = (x_1 \vee \neg x_2 \vee x_3)$ be a clause of a CNF formula $\varphi$, 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

[1] Observe that all the assignments made before the first decision assignment correspond to decision level 0.

implied variable assignments, as well as a description of mechanisms for their identification, can be found for example in [13].

## 3. SAT algorithms

Over the years a large number of algorithms have been proposed for SAT, from the original Davis-Putnam procedure [5], to recent backtrack search algorithms [1,3,9,13,14,21], to local search algorithms [17], among many others.

SAT algorithms can be characterized as being either *complete* or *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. (Observe that many of these instances are unsatisfiable, and that non-systematic algorithms cannot prove unsatisfiability.) This belief has been amply supported by extensive experimental evidence obtained in recent years [2,3,9,13,14].

In the next subsection we describe backtrack search algorithms for SAT. Moreover, we will also explain how randomization has been utilized in SAT algorithms. These two issues will motivate the stochastic systematic search algorithm, to be described in Section 4.

### 3.1. Backtrack search

The vast majority of backtrack search SAT algorithms build upon the original backtrack search algorithm of Davis, et al. [4]. A generic organization of backtrack search for SAT is composed of three main engines:

- The *decision engine* which selects a variable assignment each time it is called.
- The *deduction engine* which applies BCP, given the current variable assignments and the most recent decision assignment.
- The *diagnosis engine* which identifies the causes of a given conflicting partial assignment.

Basically, there are three main approaches for improving backtrack search SAT algorithms. The first approach consists of fine-tuning the implementation details, being generally accepted that fast implementations of SAT algorithms can yield important performance improvements [6,9,14]. The second approach for improving a SAT algorithm is the procedure for selecting decision assignments, which in most cases has a very significant impact on the overall efficiency of the algorithm [9,14], and as a result a large number of decision making heuristics for SAT has been proposed over the years. Finally, the third approach for improving backtrack search algorithms consists of reducing the space that must actually be searched, by utilizing different search pruning techniques. Experimental results [13,3] strongly suggest that pruning the search can be extremely effective for many classes of instances of SAT.

Recent state-of-the-art backtrack search SAT solvers [3,9,13,14,21] utilize sophisticated variable selection heuristics, implement fast BCP 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. Clauses that are recorded due to diagnosing conflicting conditions are referred to as *conflict-induced clauses* (or simply *conflict clauses*).

### 3.2. 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 [17]. Indeed, 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 [12].

Randomization has also been successfully included in variable selection heuristics of backtrack search algorithms [3]. Variable selection heuristics, by being greedy in nature, are bound to select the wrong variable at the wrong time for the wrong instance. The utilization of randomization helps reducing the likelihood of seeing this happening.

Although intimately related with randomizing variable selection heuristics, randomization is also a key aspect of search restart strategies [10]. Search restarts are a well-known strategy for coping with hard real-world satisfiable (and often unsatisfiable) instances, that have been integrated in recent SAT solvers [2,9,13,14]. In search restarts, randomization ensures with high probability that different sub-trees are searched each time the search algorithm is restarted.

Current state-of-the-art SAT solvers already incorporate the above forms of randomization [2,9,14]. In these solvers, variable selection heuristics are randomized and search restart strategies are utilized.

## 4. Stochastic systematic search

In this section we describe how randomization can be used within backtrack search algorithms to yield a *stochastic systematic search* SAT algorithm.

As previously described in Section 3.1, a backtrack search algorithm can be organized according to three main engines: the decision engine, the deduction engine and the diagnosis engine. Given this organization, we define a backtrack search (and so systematic) SAT algorithm to be *fully stochastic* provided all three engines are subject to randomization.

The introduction of randomization in each of the three main engines is illustrated in Fig. 1.

(1) Randomization can be (and has actually been [2,3,9,14]) applied to the decision engine by randomizing the variable selection heuristic.
(2) Randomization can be applied to the deduction engine by randomly picking the order in which implied variable assignments are handled during BCP.
(3) The diagnosis engine can be randomized by randomly selecting the point to backtrack to.

Regarding the deduction engine, observe that the number of implied variable assignments does not depend on the introduction of randomization (assuming that no conflicts are identified). For this engine, randomization only affects the order in which assignments are implied, and hence can only affect which conflict clause is identified first. Even though randomizing the deduction engine allows randomizing which conflicting clause is analyzed each time a conflict is identified, the number of conflicting clauses is in general not large, and so it is not clear whether randomization of the deduction engine can play a significant role. As a result, we chose to randomize only the two other engines of the backtrack search SAT algorithm.

**Algorithm 1.** RANDOMIZED_BACKTRACKING($\omega_c$)

$\omega$= Record_conflict_clause($\omega_c$);
*Randomly select decision assignment variable* $v_x$ *in* $\omega$;
Apply_Backtrack_Step ($v_x$, $\omega$);

Since the randomization of the decision engine is simply obtained by randomizing the variable selection heuristic [2,3,9,14], we focus on the randomization of the diagnosis engine. Moreover, by noting that the diagnosis engine defines where to backtrack to each time a conflict is identified, randomization of this engine can be achieved by randomly selecting the backtrack point. In the next paragraphs, we describe how this can actually be done.

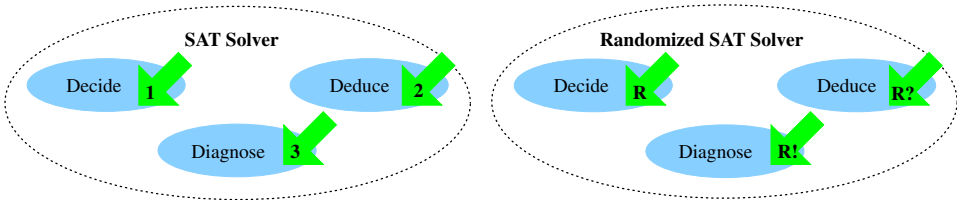


Fig. 1. Introducing randomization in a SAT solver.

As mentioned above, the most simple form of backtracking is chronological backtracking, for which each backtrack step is always taken to the most recent yet untoggled decision assignment. Moreover, state-of-the-art SAT solvers currently utilize different forms of non-chronological backtracking, for which each identified conflict is analyzed, its causes identified, and a new clause created and added to the CNF formula. Created clauses are then used to compute the backtrack point as the *most recent* decision assignment from all the decision assignments represented in the recorded clause. Backtracking to this *most recent* decision assignment ensures completeness even when recorded clauses are eventually deleted [13].

Alternatively, we can randomly pick the backtrack point from the set of literals in each recorded conflict clause. This form of backtracking is referred to as *random backtracking*. The randomized backtracking procedure is outlined in Algorithm 1. After a conflict (i.e. an unsatisfied clause $\omega_C$) is identified, a *conflict clause* $\omega$ is created. The conflict clause is then used for *randomly* deciding which decision assignment variable $v_x$ is to be toggled. This contrasts with the usual non-chronological backtracking approach, in which the most recent decision assignment variable is selected as the backtrack point.

## 5. Unrestricted backtracking

In recent years, different backtrack search algorithms for SAT have proposed backtracking relaxations, that entail relaxing the identification of the backtrack point in the search tree. Examples of backtracking relaxations include search restarts [10] and random backtracking. Observe that these kinds of strategies are currently used in some of the most effective state-of-the-art SAT solvers [9,14]. As a result, a new backtrack search strategy, UB, that naturally captures existing backtracking relaxations, has recently been proposed [11].

Even though backtracking relaxations can be significant in solving hard instances of SAT, it is also true that the resulting algorithms may no longer be *complete*. Nevertheless, as has also been shown in [11], it is possible to establish simple completeness conditions.

### 5.1. The UB algorithm

UB relaxes the condition that backtracking must be taken to the *most recent* decision assignment in a recorded clause. In other words, whenever a search conflict is reached, the search algorithm is allowed to *unrestrictedly* backtrack to *any* point (i.e. decision level) in the current search path. Clearly, this unrestricted backtrack step can also be the usual chronological or non-chronological backtrack steps. Besides the freedom for selecting the backtrack point in the decision tree, UB entails a *policy* for applying different backtrack steps in sequence. Each backtrack step can be selected among chronological backtracking (CB), non-chronological backtracking (NCB) or incomplete forms of backtracking (IFB) (e.g. search restarts [10], weak-commitment search [19], random backtracking, among many others). More formally, UB consists of defining a sequence of backtrack steps $\{BSt_1, BSt_2, BSt_3, \ldots\}$ such that each backtrack step $BSt_i$ can either be a chronological, a non-chronological or an IFB, i.e. $BSt_i \in \{CB, NCB, IFB\}$.

The definition of UB allows capturing the backtracking search strategies used by current state-of-the-art SAT solvers [3,9,13,14]. Clearly, if the UB strategy specifies always applying the CB step or always applying the NCB step, then we respectively capture the chronological and non-chronological backtracking search strategies. It is straightforward to capture other backtracking search strategies. For example, consider a UB strategy consisting of applying an IFB step after every $K$ chronological backtrack steps, and where the IFB step is a search restart. Then this UB strategy corresponds to search restarts in (chronological) backtrack search [10],[2] hence being an *incomplete* algorithm. If instead of chronological backtracking, non-chronological backtracking with clause recording is applied in between IFB steps, and if the number of conflicts in between IFB steps *increases* by a constant value $i_K$, then the resulting UB strategy corresponds to search restarts with non-chronological backtrack search (and with clause recording) [2]. Finally, if instead of a search restart, the IFB step in the previous strategies consists of a random backtrack step, then the resulting algorithm corresponds to stochastic systematic search (described in Section 4).

One can envision new, though more elaborate, UB strategies, that involve different forms of IFB. An example of such a UB strategy could be applying an IFB corresponding to random backtracking after every $K$ conflicts (where $K$

[2] In this case we assume that the variable branching heuristic can be randomized as described in [10].

can be strictly increasing), and applying an IFB step corresponding to a search restart after every $M$ conflicts (where $M$ can also be strictly increasing).

### 5.2. Completeness conditions

Clearly, for each new UB strategy one needs to ensure the completeness of the resulting algorithm. Given the generality of the UB strategy, this at first seems to be a challenging task. Nevertheless, as has been shown in [11], it is possible to establish simple completeness conditions:

(1) An UB algorithm is complete if it records (and keeps) a *conflict-clause* for each identified conflict for which an IFB step is taken.
(2) Given an integer constant $M$, an UB algorithm is complete if it records (and keeps) a *conflict-clause* after every $M$ identified conflicts for which an IFB step is taken.
(3) Suppose an UB strategy that applies a sequence of backtrack steps. If for this sequence the number of conflicts in between IFB steps strictly increases after each IFB step, then the resulting algorithm is complete.
(4) Suppose an UB strategy that applies a specific sequence of backtrack steps. If for this sequence, either the size of the largest recorded clause kept [13] or the size of the relevance-based learning threshold [3] is strictly increased after each IFB step is taken, then the resulting algorithm is complete.

Observe that these conditions are valid for *all* organizations of UB, independently of the actual organization of each UB strategy. Indeed, these conditions are valid for search restarts and randomized backtracking, and consequently for configurations combining both of them.

## 6. Experimental results

This section presents and analyzes experimental results that evaluate the effectiveness of the techniques proposed in this paper in solving hard real-world problem instances. Recent examples of such instances are the superscalar processor verification problem instances developed by M. Velev and R. Bryant [18]. We consider three sets of instances: *sss1.0a* with nine satisfiable instances, *sss1.0* with 40 selected satisfiable instances, and *sss-sat-1.0* with 100 satisfiable instances. For all the experimental results presented in this section a PIII @ 866 MHz Linux machine with 512 MByte of RAM was used. The CPU time limit for each instance was set to 1000 s. Since randomization was used, the number of runs was set to 100. Moreover, the results shown correspond to the median values for all the runs.

In order to analyze the different techniques, a new SAT solver—Quest0.5—has been implemented. Quest0.5 is built on top of the GRASP SAT solver [13], but incorporates search restarts as well as *random backtracking*.

Moreover, for the experimental results shown below, the following configurations were selected:

- Ncb indicates that only non-chronological backtracking is applied.
- Rst1000 indicates that restarts are applied after every 1000 conflicts. (Ncb is applied in the remaining steps.)
- RB1 indicates that random backtracking is taken at every backtrack step.
- RB10 applies random backtracking after every 10 conflicts.
- Rst1000 + RB1 means that random backtracking is taken at every step, except when search restarts are applied (after every 1000 conflicts).
- Rst1000 + RB10 means that random backtracking is taken after every 10 conflicts and also that restarts are applied after every 1000 conflicts.

The results for Quest0.5 on the SSS instances are shown in Table 1. In this table, *Time* denotes the CPU time, *Nodes* the number of decision nodes, and *X* the average number of aborted problem instances. As can be observed, the results for Quest0.5 reveal interesting trends:

- Random backtracking taken at every backtrack step allows significant reductions in the number of decision nodes.

Table 1
Results for the SSS instances

| Instances | *sss1.0a* | | | *sss1.0* | | | *sss-sat-1.0* | | |
|---|---|---|---|---|---|---|---|---|---|
| *Quest 0.5* | *Time* | *Nodes* | *X*[a] | *Time* | *Nodes* | *X* | *Time* | *Nodes* | *X* |
| Ncb | 1603 | 257 126 | 8 | 2242 | 562 178 | 11 | 83 030 | 12 587 264 | 82 |
| Rst1000 | 208 | 59 511 | 0 | 508 | 188 798 | 0 | 50 512 | 8 963 643 | 39 |
| RB1 | 79 | 11 623 | 0 | 231 | 29 677 | 0 | 10 307 | 371 277 | 1 |
| RB10 | 204 | 43 609 | 0 | 278 | 81 882 | 0 | 6807 | 971 446 | 1 |
| Rst1000 + RB1 | 79 | 11 623 | 0 | 221 | 28 635 | 0 | 10 330 | 396 551 | 2 |
| **Rst1000 + RB10** | **84** | **24 538** | **0** | **147** | **56 119** | **0** | **7747** | **1 141 575** | **0** |

[a]Corresponds to the number of aborted instances.

- The best results are always obtained when random backtracking is used, independently of being or not being used together with restarts.
- **Rst1000 + RB10** is the only configuration able to solve all the instances in the allowed CPU time for *all* the runs.

The experimental results reveal additional interesting trends. When compared with the results where only non-chronological backtracking is applied, the remaining configurations (which apply search restarts and/or randomized backtracking) yield dramatic improvements. (This is confirmed by evaluating either the CPU Time, the number of nodes or the number of aborted instances.) Furthermore, even though the utilization of restarts reduces the amount of search, it is also clear that more significant reductions can be achieved with randomized backtracking. In addition, the integrated utilization of search restarts and randomized backtracking allows obtaining the best results, and therefore clearly illustrate the practical effectiveness of the proposed techniques.

## 7. Related work

The introduction of relaxations in the backtrack step is related to dynamic backtracking [7]. Dynamic backtracking establishes a method by which backtrack points can be moved deeper in the search tree. This allows avoiding the unneeded erasing of the amount of search that has been done thus far. The objective is to find a way to directly "erase" the value assigned to a variable as opposed to backtracking to it, moving the backjump variable to the end of the partial solution in order to replace its value without modifying the values of the variables that currently follow it. More recently, Ginsberg and McAllester combined local search and dynamic backtracking in an algorithm which enables arbitrary search movement [8], starting with *any complete assignment* and evolving by flipping values of variables obtained from the conflicts.

In weak-commitment search [19], the algorithm constructs a consistent partial solution, but commits to the partial solution *weakly*. In weak-commitment search, whenever a conflict is reached, the *whole* partial solution is abandoned, in explicit contrast to standard backtracking algorithms where the most recently added variable is removed from the partial solution.

Moreover, search restarts have been proposed and shown effective for hard instances of SAT [10]. The search is repeatedly restarted whenever a cutoff value is reached. The algorithm is not complete, since the restart cutoff point is kept constant. In [2], search restarts were jointly used with learning for solving hard real-world instances of SAT. This latter algorithm is complete, since the backtrack cutoff value increases after each restart. One additional example of backtracking relaxation is described in [16], which is based on attempting to construct a complete solution, that restarts each time a conflict is identified.

More recently, CLS, proposed by Prestwich [15], involves randomizing the backtracking component by allowing backtracking to *arbitrarily-chosen* variables. In CLS [15], 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, and so can be very competitive for specific classes of problem instances. We should note, however, that the CLS algorithm is not complete and so it cannot establish *unsatisfiability*.

## 8. Conclusions

This paper proposes and analyzes the application of randomization in the different components of backtrack search SAT algorithms. A new, stochastic but complete, backtrack search algorithm for SAT is proposed.

In conclusion, the main contributions of this paper can be summarized as follows:

(1) A new backtrack search SAT algorithm is proposed, that randomizes the variable selection and the backtrack steps.
(2) Randomized backtracking is naturally captured by unrestricted backtracking, a generic framework for combining different strategies for backtrack search [11]. As a result, it is possible to obtain an algorithm which applies randomized backtracking and other strategies, namely search restarts and non-chronological backtracking.
(3) The proposed SAT algorithm is shown to be complete, and different approaches for ensuring completeness are described.
(4) Experimental results clearly indicate that significant savings in the search effort can be obtained for different organizations of the proposed algorithm.

It is generally accepted that restarts are useful when the run time distribution exhibits a high variance. A question that might be interesting to address in future research is: how does adding random backtracking affect the variance (of the run time)? Understanding this relationship may allow one to identify the optimal mix of restarts and random backtracking.

Future research work will also consider other variations of this new algorithm. We can envision using the unrestricted backtracking framework for implementing backtracking strategies, allowing the implementation of different hybrids, all guaranteed to be complete and so capable of proving unsatisfiability. In this generic framework, the actual backtrack point can be defined using either randomization, constant-depth backtracking or search restarts, among other possible approaches.

## Acknowledgments

This work is partially supported by Fundação para a Ciência e Tecnologia under research projects POSI/EEI/11249/1998 and POSI/34504/CHS/2000.

## References

[1] F. Bacchus, Exploiting the computational tradeoff of more reasoning and less searching, in: Fifth International Symposium on Theory and Applications of Satisfiability Testing, 2002, pp. 7–16.
[2] L. Baptista, J.P. Marques-Silva, Using randomization and learning to solve hard real-world instances of satisfiability, in: R. Dechter (Ed.), International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, vol. 1894, Springer, Berlin, 2000, pp. 489–494.
[3] R. Bayardo Jr., R. Schrag, Using CSP look-back techniques to solve real-world SAT instances, in: Proceedings of the National Conference on Artificial Intelligence, 1997, pp. 203–208.
[4] M. Davis, G. Logemann, D. Loveland, A machine program for theorem-proving, Commun. Assoc. Comput. Mach. 5 (1962) 394–397.
[5] M. Davis, H. Putnam, A computing procedure for quantification theory, J. Assoc. Comput. Mach. 7 (1960) 201–215.
[6] J.W. Freeman, Improvements to propositional satisfiability search algorithms, Ph.D. Thesis, University of Pennsylvania, Philadelphia, PA, May 1995.
[7] M. Ginsberg, Dynamic backtracking, J. Artif. Intell. Res. 1 (1993) 25–46.
[8] M. Ginsberg, D. McAllester, GSAT and dynamic backtracking, in: Proceedings of the International Conference on Principles of Knowledge and Reasoning, 1994, pp. 226–237.
[9] E. Goldberg, Y. Novikov, BerkMin: a fast and robust sat-solver, in: Proceedings of the Design and Test in Europe Conference, 2002, pp. 142–149.
[10] C.P. Gomes, B. Selman, H. Kautz, Boosting combinatorial search through randomization, in: Proceedings of the National Conference on Artificial Intelligence, 1998, pp. 431–437.
[11] I. Lynce, J. Marques-Silva, Complete unrestricted backtracking algorithms for satisfiability, in: Fifth International Symposium on the Theory and Applications of Satisfiability Testing, 2002, pp. 214–221.
[12] D. McAllester, B. Selman, H. Kautz, Evidence of invariants in local search, in: Proceedings of the National Conference on Artificial Intelligence, 1997, pp. 321–326.
[13] J.P. Marques-Silva, K.A. Sakallah, GRASP—A search algorithm for propositional satisfiability, IEEE Trans. Comput. 48 (5) (1999) 506–521.
[14] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, Engineering an efficient SAT solver, in: Proceedings of the Design Automation Conference, 2001, pp. 530–535.

[15] 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, 2000, pp. 337–352.
[16] E.T. Richards, B. Richards, Non-systematic search and no-good learning, J. Automat. Reason. 24 (4) (2000) 483–533.
[17] B. Selman, H. Kautz, Domain-independent extensions to GSAT: Solving large structured satisfiability problems, in: Proceedings of the International Joint Conference on Artificial Intelligence, 1993, pp. 290–295.
[18] M.N. Velev, 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, Lecture Notes in Computer Science, vol. 1703, 1999, pp. 37–53.
[19] M. Yokoo, Weak-commitment search for solving satisfaction problems, in: Proceedings of the National Conference on Artificial Intelligence, 1994, pp. 313–318.
[20] R. Zabih, D.A. McAllester, A rearrangement search strategy for determining propositional satisfiability, in: Proceedings of the National Conference on Artificial Intelligence, 1988, pp. 155–160.
[21] H. Zhang, SATO: An efficient propositional prover, in: Proceedings of the International Conference on Automated Deduction, 1997, pp. 272–275.