

On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search

(Extended Abstract*)

Ryan Williams¹, Carla Gomes², and Bart Selman²

¹ Dept. of Comp. Science, CMU, Pittsburgh, PA 15213-3891
ryanw@cs.cmu.edu

² Dept. of Comp. Science, Cornell University, Ithaca, NY 14853
{gomes, selman}@cs.cornell.edu

1 Introduction

Recent state-of-the-art SAT solvers can handle hand-crafted instances with hundreds of thousands of variables and several million clauses. Only a few years ago, the ability to handle such instances appeared completely out of reach. The most effective complete solvers are generally based on Davis-Putnam-Loveland-Logemann style search procedures augmented with a number of special techniques, such as clause-learning, non-chronological backtracking, lookahead, fast unit-propagation, randomization, and restart strategies. The progress in this area has largely been driven by experimental work on diverse sets of benchmark problems, including regular SAT competitions. Given the tremendous advances in recent years, this has clearly been a highly successful approach. One key open area of research is to obtain a better understanding as to *why* these methods work so well. In this paper, we hope to advance our understanding of the effectiveness of current techniques and analyze what features of practical instances makes them so amenable to these solution methods. Of the many enhancements of DPLL, we will focus our attention on the interplay between certain special features of problem instances, polytime propagation methods, and restart techniques. This analysis is clearly only part of the full story, since other enhancements, such as clause learning and non-chronological backtracking, provide additional power to these solvers.

In order to characterize hidden structure in problem instances, we introduced a notion of a special subset of variables, called the *backdoor* variables [21]. A set of variables forms a backdoor for a problem instance if there is a value assignment to these variables such that the simplified formula can be solved in polynomial time by the propagation and simplification mechanism of the SAT solver under consideration. Another way of stating this is to say that after setting the backdoor variables the simplified formula falls in a polynomially solvable class. Note however that this class may not have a simply syntactic characterization.

There is no apriori reason why backdoor sets are interesting. However, we have observed that structured problem instances can have surprisingly small sets of backdoor variables. For example, the logistics-d planning problem instance (log.d.cnf) has a backdoor set of just 12 variables, compared to a total of nearly 7,000 variables in the formula. More specifically, there is a setting to these variables, such that the remaining formula is solved immediately by the polytime propagation techniques of the SAT solver, satz [14]. We have found similarly small backdoors for other structured problems instances, such as those from bounded model-checking domains. Moreover, we have also identified classes of formulas that do not appear to have small backdoors. For example, for random 3SAT problems, the backdoor appears to be a constant fraction (roughly 30%) of the total number of variables [11]. This may explain why the current DPLL based solvers have not made significant progress on hard randomly

* This paper complements the work reported in [21]. In [21] we formally introduced the notion of backdoor and characterize the complexity of several algorithms designed to take advantage of backdoors, including restart strategies. In this paper, we focus on the connection between heavy-tails and backdoors.

generated instances. (Hard random 3-SAT instances with more than 400 variables are out of reach of most complete solvers. See also analysis in [1].)

Backdoor variables are related to the notion of independent variables [20, 5]. In fact, a set independent variables of a problem also forms a backdoor set (provided the propagation mechanism of the SAT solver, after setting the independent variables, can effectively uncover the remaining variable dependencies). In practice, backdoors can still be quite a bit smaller than the set of independent variables. For example, in the logistics planning domain, the set of independent variables is given by the number of operators applicable at each time step. This set of variables will generally be much larger than the minimal backdoor set of 12 variables, mentioned earlier. Intuitively, the minimal backdoor also incorporates a notion of *critical variables*, i.e., those variables that capture some of the critically constrained resources of the original problem.¹

Once a (small) backdoor set has been identified, the SAT solver can restrict its search to the setting of those variables, leading to a $2^k \text{poly}(n)$ search for a backdoor with k variables, and $\text{poly}(n)$ capturing the polynomial propagation after setting the backdoor variables. For small k , in particular of size $O(\text{Log}(n))$, these leads to a very effective overall search. Of course, in practice, the small backdoor set (if it exists) still needs to be uncovered by the SAT solver itself. In [21], we show that even when taking into account the cost of searching for backdoor variables, one can still obtain an overall computational advantage by focusing in on a backdoor set, provided the set is sufficiently small. Heuristics, incorporated in many in current SAT solvers, also implicitly search for backdoor variables, by uncovering those variables that cause a large number of unit-propagation.

The notion of backdoor came about in the context of our study of heavy-tailed behavior as observed in backtrack style search [10, 9]. Heavy-tailed distributions provide a justification of why restarts are effective, namely to prevent the search procedure from getting stuck in un-productive portions of the search space that do not contain solutions. Such distributions also imply the existence of a wide range of solution times, often including short runs. For example, in [4], we proposed a formal model of heavy-tailed behavior for combinatorial search characterized by an infinite mean, yet, with a reasonable probability of finding solutions by searching small sub-trees of sizes $2, 2^1, 2^2, \dots, 2^k$ nodes (for small k). This is where backdoors come in. Intuitively, a small backdoor explains how a backtrack search can get “lucky” on certain runs, where backdoor variables are identified early on in the search and set the right way. (Below we also introduce a notion of “strong backdoor” which explains short, unsatisfiable runs.) In fact, we can now provide a detailed mathematical model explaining heavy-tailed behavior (Pareto like tails) in backtrack search as function of the size of the minimal backdoor set.

Several researchers have demonstrated the effectiveness of randomization and restart strategies for solving real-world applications. For example, the work of Marques-Silva *et al.* has shown the success of such techniques, combined with learning, for Electronic Design Automation ([18, 2, 16]). Randomization and restart strategies are now an integral part of many current SAT solvers ([3, 8, 13, 16, 19, 23]).²

The structure of the paper is as follows. We start by formally introducing the concept of backdoor and provide empirical results on the size of the backdoor. We then relate the backdoor notion to variable choice trees and provide formal heavy-tailedness results as a function of the size of the backdoor.

¹ For certain problem classes, such as parity problems and DES instances, the minimal backdoor set coincides with the full set of independent variables, forcing a backtrack search procedure to exhaustively search the setting of these variables [7].

² A restart strategy does not need to involve explicit randomization. For example, the clause learning mechanism in Chaff, implicitly causes the solver to explore different parts of the search space on each restart (a form of “deterministic randomization” [22]).

2 Backdoors

In this section, we define the notion of backdoor variables, first introduced in [21]. We will also present some of the empirical results for backdoors. Our definitions are for the SAT problem but can easily be adapted to the more general setting of constraint satisfaction problems (CSP).

The SAT problem is defined in the usual way. We have a Boolean formula over n Boolean variables. The formula consists of a conjunction of clauses, where each clause consist of a disjunction of literals. A literal is a Boolean variable or its negation. SAT is the problem of determining whether there exists a truth assignment to the Boolean variables that satisfies all clauses.

Let $a_S : S \subseteq V \rightarrow \{True, False\}$ be a truth partial assignment. We use $F[a_S]$ to denote the simplified SAT instance obtained from the formula F by setting the variables defined in a_S , i.e., by fixing the truth values of the variables in a_S .

A set of backdoor variables is defined with respect to a particular algorithm; once the backdoor variables are assigned a value, the problem becomes easy under that algorithm. To begin our exposition of backdoors, we define the sort of algorithms we have in mind. We will call them *sub-solvers*, as they solve tractable subcases of the general SAT problem.

Definition 1. *A sub-solver A given as input a formula F satisfies the following:*

- (Trichotomy) *A either rejects the input F , or “determines” F correctly (as unsatisfiable or satisfiable, returning a solution if satisfiable),*
- (Efficiency) *A runs in polynomial time,*
- (Trivial solvability) *A can determine if F is trivially true (has no clauses) or trivially unsatisfiable (has a contradictory clause),*
- (Self-reducibility) *if A determines F , then for any variable x and value v , then A determines $F[v/x]$.*

For instance, A could be an algorithm that solves 2-SAT instances but rejects all other instances. In general, A captures the polytime propagation and simplification mechanisms that are part of DPLL style SAT solvers. Our results below are independent of a particular sub-solver; our results will hold for any A satisfying the above four properties.

In what follows, let A be a sub-solver, and F a Boolean formula.

We first consider a notion of “backdoor” that is suitable for satisfiable problems instances.

Definition 2. [backdoor] *A nonempty subset S of the variables is a backdoor for F w.r.t. A if for some $a_S : S \rightarrow \{False, True\}$, A returns a satisfying assignment of $F[a_S]$.*

Intuitively, the backdoor corresponds to a set of variables, such that when set correctly, the sub-solver can solve the remaining problem. In a sense, the backdoor is a “witness” to the satisfiability of the instance, given a sub-solver algorithm. We also introduce a stronger notion of the backdoor to deal with both satisfiable and unsatisfiable (inconsistent) problem instances.

Definition 3. [strong backdoor] *A nonempty subset S of the variables is a strong backdoor for C w.r.t. A if for all $a_S : S \rightarrow \{False, True\}$, A returns a satisfying assignment or concludes unsatisfiability of $F[a_S]$.*

It follows directly that, when given the backdoor S for a SAT problem, the search cost is of order $\text{poly}(n)2^{|S|}$. (Simply check all possible assignments of S .) This means if S is relatively small, one obtains a large improvement over searching the full space of variable/value assignments.

We observe that *independent* variables are a particular kind of backdoor. As stated in [12], they are a set S of variables for which all other variables may be thought of as *defined* in terms of S . For example, a maximal subset of independent variables in a SAT encoding of a hardware verification problem is a backdoor for unit propagation, as the other variables’ values may be directly determined after setting the independent ones [17, 20, 5].

There are two key questions concerning backdoors: (1) What is the size of the backdoor in practical problem instances? (2) When taking into account the cost of searching for a backdoor set, can one still obtain an overall computational advantage?

In [21], we present formal complexity results that show a concrete computational advantage of using backdoor variables even when taking into account the cost of searching for the backdoor set, provided that a relatively small backdoor set exists. Moreover, empirical data shows that for practical structured problem instances the backdoor sets can indeed be surprisingly small. Table 1 gives some example statistics on backdoor sizes. The backdoors in the table were obtained using the Satz SAT solver SAT solver Satz-rand, a randomized version of Satz [14]. Satz incorporates powerful variable selection heuristics and an efficient simplification strategy (*i.e.*, a good sub-solver). In the table, we show a logistics planning problem (log.d), a circuit design problem (3bitadd_32), a bounded model-checking problem (pipe), and two quasigroup completion problems.

We see some surprisingly small backdoor sets. Clearly, the problem instances have lots of hidden structure and moreover such structure can be uncovered by the propagation and simplification procedures in Satz.³

It should be noted that these instances are now well within the range of the fastest current solvers, such as Satz [14]. However, they are non-trivial and cannot be solved with the previous generation of SAT solvers, such as Tableau [6]. Clearly, the new solvers are better able to discover and exploit hidden structure, such as small backdoors.

instance	# vars	# clauses	backdoor	fract.
log.d	6783	437431	12	0.0018
3bitadd_32	8704	32316	53	0.0061
pipe_01	7736	26087	23	0.0030
qwh_30_320	1235	8523	14	0.0113
qwh_35_405	1597	10658	15	0.0094

Table 1. Size of backdoors for several practical SAT instances. Source: [21]

As mentioned earlier, the notion of backdoor set came about when we studied the mechanisms underlying heavy-tailed phenomena in combinatorial search. In the next section, we provide a formal analysis making the connection between backdoors, heavy-tails and restarts precise.

3 Search Trees and Backdoors

For our analysis, we introduce the notion of *variable choice trees*, binary trees with integer labels on their leaves, but with semantics quite different from standard search trees. The typical semantics given

³ The table gives results on satisfiable instances. We believe similarly small strong backdoors exist in many structure unsatisfiable problem instances. We are currently modifying satz-rand to allow us to search for such strong backdoors.

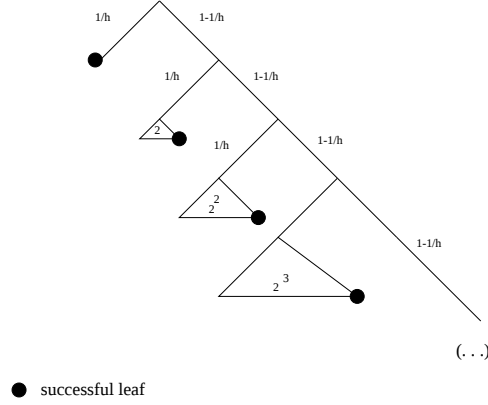


Fig. 1. Variable choice tree with one backdoor; $1/h$ is the probability of the branching heuristic making a good decision and b is the branching factor, in this case $b = 2$. This tree model has finite mean and infinite variance when $1/2 < 1/h < 3/4$, and infinite variance and infinite mean when $1/h \leq 1/2$

to a search tree is that the nodes represent variables, and the branches out of a node represent the various values that can be assigned to node variable. In variable choice trees, an inner node represents the *selection of a variable*, so branches correspond to different variable choices, and the integer label of leaf l represents the cost of a search on the variables that were selected in the path to l . In our work, we consider the variables of an instance to be partitioned into two kinds; for the purposes of abstraction we will simply call the two partitions “good” and “bad” here. Intuitively, good variables are ones that we would like to choose early in our search; for us, these will be backdoor variables. As we choose them earlier in the search, the solver performs better. Left-branches of the tree represent good variable choices, and right-branches represent bad ones. Once all of the good (backdoor) variables have been chosen in a path of the selection tree, the path ends with a leaf labeled with the search cost of an algorithm run on those variable choices described by the branches. Following the empirical work in [9], where randomization of variable selection leads to increased performance in search, we attach probabilities to the branches, so that good variables are chosen with some probability $1/h$ and bad ones with $1 - 1/h$. Figure 1 depicts a variable choice tree when we have a backdoor consisting of a single variable.

Notice that the performance of several deterministic search algorithms for SAT/CSPs could be represented with one variable choice tree, as for each problem and each algorithm there is a unique ordering of the variables chosen for branching, which corresponds to different paths in the variable choice tree. In order to formalize the notion of variable choice tree, we will use a recursive characterization. This version illuminates the fact that a variable choice tree specifies a kind of tree that is self-similar, i.e., fractal.

Definition 4. $V(B)$ - variable choice tree for B backdoor variables (minimum-sized); *Recursive formulation.*

1. $V(0)$ is a single leaf.
2. For $B > 0$, $V(B)$ has $V(B - 1)$ as a left subtree, and a copy of $V(B)$ as a right subtree.

For example, $V(1)$ is the infinite binary tree in Figure 1. See Figure 2 for $V(2)$ and $V(3)$.

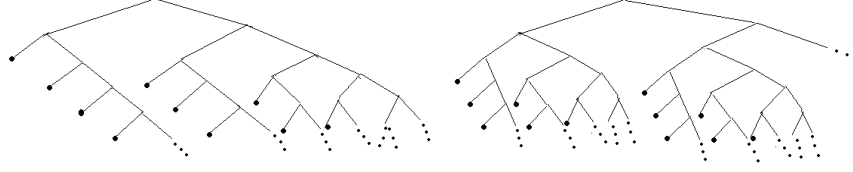


Fig. 2. Sketch of $V(2)$ and $V(3)$.

We augment the tree $V(B)$ with probability labels on its edges and search cost labels on the leaves, as informally suggested above. To simplify the exposition, throughout the paper we will assume the edges of $V(B)$ are labeled with probabilities; the left and right branches will have labels $1/h$ and $1 - 1/h$, respectively. The leaves of $V(B)$ will be labeled according to a *search cost* function $f : \mathbb{N} \rightarrow \mathbb{N}$, which is monotone increasing. Leaf l gets label $f(d(l))$, where $d(n)$ is the depth of node n . The function f is meant to model the impact of choosing good variables over bad ones, and vice-versa.

4 On the connections between heavy-tailedness and backdoors

Here, we outline our formal results connecting a backtrack search heuristic search model with heavy-tailed runtime phenomena. In particular, we will show that small sets of backdoor variables lead to runtime profiles that are bounded from below by heavy-tails.

Throughout, we fix a particular variable choice tree $V(B)$, search cost function f , and heuristic having success probability $1/h$. Our first step is to define our notation for the tail probability of $V(B)$.

Definition 5. $Pr[t \geq L \mid V(B)]$, the *tail probability of L* , is the probability that a randomly chosen path (to some leaf l of $V(B)$) has search cost $f(l) \geq L$.

Roughly speaking, the tail probability of L is the probability that random variable choices under the heuristic result in a runtime greater than L . In the following we assume $f(k) = b^k$ for some $b > 1$. (b is the branching factor. $b = 2$ for SAT.) That is, the search cost at a leaf is exponential in length of the path leading to the leaf. Hence for $B = 1$, $Pr[t \geq b^{i+1} \mid V(B)] = (1 - 1/h)^i$; this result was essentially shown in [4]. Here we generalize that result. A lemma gives a recursive expression for the probability when $B > 1$.

Lemma 1. For $i > 0$ and $B > 1$,

$$Pr[t \geq b^{i+B} \mid V(B)] = \frac{1}{h} \cdot Pr[t \geq 2^{i+(B-1)} \mid V(B-1)] + \left(1 - \frac{1}{h}\right) \cdot Pr[t \geq b^{(i-1)+B} \mid V(B)].$$

Proof. Omitted due to lack of space - to be included in the proceedings version.

4.1 The tail probability theorem

Our next step is to find a closed-form solution for this tail probability. We first give the solution when $B = 2$ as a warm-up to the theorem for arbitrary B .

Theorem 1. $Pr[t \geq b^{i+2} \mid V(2)] = (1 - 1/h)^i (1 + i/h)$.

Proof. Omitted due to lack of space - to be included in the proceedings version.

Theorem 2. For all $i \geq 1$,

$$Pr[t \geq b^{i+B} \mid V(g)] = (1 - \frac{1}{h})^i (1 + \sum_{k=1}^{B-1} \binom{i+k-1}{k} \cdot \frac{1}{h^k}).$$

Proof. Omitted due to lack of space - to be included in the proceedings version.

Now we define what we mean by $V(B)$ having a heavy-tailed distribution. Our only concern is with having a search cost distribution that is lower bounded by a heavy-tail; if the tail is *larger* than Pareto-Levy, then a restart strategy is only more viable.

Definition 6. $V(B)$ is lower bounded by a heavy-tail if there exists $\alpha \in (0, 2)$ such that $Pr[t \geq L \mid V(B)] = \Omega(\frac{1}{L^\alpha})$. That is, the search cost distribution on $V(B)$ is bounded from below by a Pareto-Levy distribution.

Note that the existence of such a heavy-tail bound does not immediately imply that a polynomial time restart strategy is available. Indeed, depending on α , the mean of the distribution could still be exponential [4]. This leads us to believe that heavy-tails may also arise for much larger backdoors than those we saw yielding efficient restart strategies earlier. Our intuition is confirmed by a calculation:

Theorem 3. (Heavy-tail lower bound) If $B \in o(N/\log N)$ and the probability of heuristic failure $(1 - 1/h)$ is less than $1/b^\alpha$ for $\alpha \in (0, 2)$, then the tail probability of the search cost on $V(B)$ is lower bounded by a heavy-tail.

Proof. Omitted due to lack of space - to be included in the proceedings version.

Intuitively, this result shows that if the success probability of the heuristic is sufficiently bad with respect to the branching factor, then there is a large potential for reaching a node in the variable choice tree $V(B)$ with high search cost.

Now let us discuss what bearing these results on variable choice trees have on randomized backtracking. Consider a model of randomized depth-first search (DFS) with a sub-solver A , running on an instance C , having a backdoor of size B , armed with a heuristic H , that chooses a variable from a minimal strong backdoor with probability $1/h$, and then randomly chooses an arbitrary assignment to the given variable $1/d$ (d is the variable domain size). We will use the notation $(\mathbf{DFS}, H, A)$ to denote a solver with the above properties. In such a model, the worst-case search cost at a leaf of the corresponding variable choice tree will be $f(k) = d^k$ (recall that k here is the depth of the variable choice tree, i.e. the number of variables chosen before a complete backdoor is chosen), simply because backtracking may have to try all possible assignments on the strong backdoor before the sub-solver can decide the instance. Therefore our above discussion concerning runs of length b^{i+B} apply directly to measuring the runtime cost of backtracking with heuristics, by setting $b = d$. We conclude with the following:

Theorem 4. $(\mathbf{DFS}, A, H)$ on SAT/CSPs with $o(N/\log N)$ strong backdoors has a runtime distribution that is lower-bounded by a heavy-tail, if the success probability of H is constant.

Proof. Omitted due to lack of space - to be included in the proceedings version.

Open problem

We believe a result of the following form is possible. Define the *restart speed-up* of a solver S on a set of instances I to be the ratio:

$$\min_{x \in I} \frac{\text{expected runtime of } S \text{ on } x}{\text{optimal runtime of any restart strategy for } S \text{ on } x}.$$

Consider the denominator of this expression. It is known that for any particular S and x , the optimal restart strategy is actually a fixed value [15]. But it is not clear how for arbitrary x how to efficiently determine what the optimal restart strategy might be. This becomes an interesting theoretical open problem when we are armed with abstract knowledge of what our solver is like (e.g. it is a randomized backtracking procedure).

In a sense, this ratio would measure the worst-case performance of a restart strategy for a particular solver, by pitting its expected runtime against the strategy. Observe that if the optimal restart strategy is not very good, the restart speed-up is close to 1, and if there is an extremely good strategy but the expectation is high, the restart speed-up can be exponential. We conjecture (for suitable definitions of the denominator) that the restart speed-up is exponential precisely *when the runtime distribution of S is lower bounded by a heavy-tail*, for infinitely many $x \in I$. Tools from Luby et al. [15] can probably be used towards proving or disproving this conjecture.

5 Conclusions

We propose a general scenario that provides valuable insights into the structure of real world instances and the way state-of-the-art procedures solve such instances. Experimentally, it appears that backdoors can be surprisingly small ($O(\log(n))$) for certain structured problems. Given the conditions of our heavy-tail lower bound result (success probability of the heuristic is sufficiently low and the backdoor size is sufficiently small), the *expected* search cost on the search tree ($V(B)$) will be exponential. Furthermore, when the size of the backdoor $B = O(\log n)$ and $1/h$ is bounded by a constant, there exists a polytime restart strategy[21]. Thus we have reasonable conditions under our model for which backtrack search has *exponential* expected runtime, yet there is a polynomial time restart strategy. In other words, we show that with a sufficiently small backdoor set and a reasonably, but not perfect, heuristic, we obtain a backtrack search with polytime restart strategy. However, without restarts, the underlying distribution is heavy-tailed, with an exponential expected time. This scenario appears to be quite close to what one observes on structured practical instances. We therefore believe that our model captures some of the key features of practical DPLL style SAT solvers. In essence, we have a subtle interplay between a set of critical variables (backdoors) and the heuristics that attempt to guide the search. On the other hand, there are instances that do not have small backdoors, such as hard random instances or instances based on cryptographic protocols. Such instances appear to be inherently hard for backtrack search and defy the current state-of-the-art complete SAT solvers. Of course, other strategies such as clause learning may yet prove to be effective on such instances.

Overall we have shown theoretical and experimental connections between the existence of backdoor sets and heavy-tailed runtimes. We hope that our work will inspire further investigation into these notions, leading to the discovery of new properties and relationships between solvers and instances.

References

1. D. Achlioptas, P. Beame, and M. S. O. Molloy. A sharp threshold in proof complexity. In *ACM Symposium on Theory of Computing*, pages 337–346, 2001.

2. L. Baptista, I. Lynce, and J. Marques-Silva. Complete restart strategies for satisfiability. In *Proceedings of the Workshop on Stochastic Search Algorithms (IJCAI)*, 2001.
3. R. Bayardo and R. Schrag. Using csp look-back techniques to solve real-world sat instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 203–208, New Providence, RI, 1997. AAAI Press.
4. H. Chen, C. Gomes, and B. Selman. Formal models of heavy-tailed behavior in combinatorial search. In *Proceedings of 7th Intl. Conference on the Principles and Practice of Constraint Programming (CP-2001)*, *Lecture Notes in Computer Science*, Vol. 2239, Springer-Verlag, pages 408–422, 2001.
5. F. Copt, L. Fix, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Vardi. Benefits of bounded model checking at an industrial setting. In *Proc. 13th Conf. on Computer Aided Verification (CAV’01)*, 2001.
6. J. Crawford and L. Auton. Experimental Results on the Crossover Point in Random 3SAT. *Artificial Intelligence*, 81(1–2), 1996.
7. E. Giunchiglia. Personal communications, 2003.
8. E. Goldberg and Y. Novikov. Berkmin: A fast and robust sat solve. In *Proc. of Design Automation and Test in Europe (DATE-2002)*, pages 142–149, 2002.
9. C. Gomes, B. Selman, and H. Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–438, New Providence, RI, 1998. AAAI Press.
10. C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1–2):67–100, 2000.
11. Y. Interian. Manuscript submitted to Sat 2003, 2003.
12. H. Kautz, D. McAllester, and B. Selman. Exploiting variable dependency in local search. In *Proceedings of the International Joint Conference on Artificial Intelligence*. AAAI Press, 1997.
13. C. M. Li. A constrained-based approach to narrow search trees for satisfiability. *Information processing letters*, 71:75–80, 1999.
14. C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.
15. M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Process. Lett.*, pages 173–180, 1993.
16. I. Lynce and J. Marques-Silva. Building state-of-the-art sat solvers. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2002.
17. J. Marques-Silva. Search algorithms for satisfiability problems in combinatorial switching circuits. PhD Thesis, Dept. of EECS, U. Michigan, 1995.
18. J. Marques-Silva and A. Sakallah. Boolean satisfiability in electronic design automation. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC-2000)*, 2000.
19. M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535, 2001.
20. B. Selman, H. A. Kautz, and D. A. McAllester. Ten challenges in propositional reasoning and search. In *Proceedings of the International Joint Conference on Artificial Intelligence*. AAAI Press, 1997.
21. R. Williams, C. Gomes, and B. Selman. Manuscript under review for Ijcai-03, 2003.
22. S. Wolfram. *A New Kind of Science*. Stephen Wolfram, 2002.
23. H. Zhang. A random jump strategy for combinatorial search. In *Proc. of International Symposium on AI and Math*, 2002.