Boosting Combinatorial Search Through Randomization

Carla P. Gomes[*]
Computer Science Department
Cornell University
Ithaca, NY 14853
gomes@cs.cornell.edu

Bart Selman
Computer Science Department
Cornell University
Ithaca, NY 14853
selman@cs.cornell.edu

Henry Kautz
AT&T Labs
180 Park Avenue
Florham Park, NJ 07932
kautz@research.att.com

Abstract

Unpredictability in the running time of complete search procedures can often be explained by the phenomenon of "heavy-tailed cost distributions", meaning that at any time during the experiment there is a non-negligible probability of hitting a problem that requires exponentially more time to solve than any that has been encountered before (Gomes *et al.* 1998a). We present a general method for introducing controlled randomization into complete search algorithms. The "boosted" search methods provably eliminate heavy-tails to the right of the median. Furthermore, they can take advantage of heavy-tails to the left of the median (that is, a non-negligible chance of very short runs) to dramatically shorten the solution time. We demonstrate speedups of several orders of magnitude for state-of-the-art complete search procedures running on hard, real-world problems.

Introduction

The time required by complete search methods to solve similar combinatorial problems can be surprisingly variable. Two problem instances may be identical, except for the order in which the variables are numbered. A particular complete search algorithm may solve the first in seconds, yet require hours or days to solve the second. Even for a single problem instance, a seemingly trivial change in a detail of the search algorithm may drastically alter the solution time. In addition, in many domains there are problem instances that can be quickly solved by incomplete methods, but apparently cannot be solved by complete methods, even methods guided by powerful heuristics.

This unpredictability in the running time of a complete algorithm undermines one of the main reasons that one may choose to employ such a method, namely the desire for a guarantee that the algorithm will determine whether or not each problem instance in fact has a solution. It is desirable, therefore, to find ways to improve the robustness and predictability of these algorithms.

[*]Carla P. Gomes is also a Research Associate at the Air Force Research Laboratory, Rome, NY, USA.

This paper discusses a general technique for improving complete search methods by introducing a controlled amount of *randomization*. The technique actually takes advantage of the variability of the underlying search method in order to find solutions more quickly and with less variance in solution time. We demonstrate the effectiveness of this strategy on SAT and CSP algorithms, in the domains of logistics planning, circuit synthesis, and round-robin scheduling. Solutions times are reduced by an order of magnitude or more, and some instances are solved for the first time by a method other than local search.

We will show that the unpredictability in running times for combinatorial algorithms can often be explained by a phenomenon called a "heavy-tailed cost distribution" (Gomes *et al.* 1998a). In our preliminary experiments, we plotted the solution times for a deterministic search algorithm running on a random distribution of scheduling problem instances. We noticed that at any time during the experiment there was a non-negligible probability of hitting a problem that required exponentially more time to solve than any that had been encountered before. This so-called "heavy-tail" phenomena causes the mean solution time to increase with the length of the experiment, and to be infinite in the limit.

Previous authors have noted the occurrence of seemingly exceptionally hard problems in fixed problem distributions (Gent and Walsh 1994; Smith and Grant 1996). However, we further discovered that when a small amount of randomization was introduced into the heuristic used by the search algorithm, then, on some runs, the instances were solved quickly. Thus, the "hardness" did not reside in the instances, but rather in the combination of the instance with the details of the deterministic algorithm. When we plotted the solution times for many runs of the randomized complete algorithm (with different random seeds) on a *single* problem instance, we discovered the same heavy-tailed distribution as we had seen before on a collection of instances.

This observation led us to realize that a deterministic search algorithm can be viewed as a *single run* of a randomized algorithm. The unpredictability of deterministic,

Problem	Solver	Deterministic soln. time	Randomized mean soln. time
logistics.d	Satz	108 min	95 sec
3bit-adder-32	Satz	$>$ 24 hrs	165 sec
3bit-adder-31	Satz	$>$ 24 hrs	17 min
round-robin 14	ILOG	411 sec	250 sec
round-robin 16	ILOG	$>$ 24 hrs	1.4 hrs
round-robin 18	ILOG	$>$ 48 hrs	$\approx$ 22 hrs
block-world.d	Satz	30 min	23 min

Table 1: Comparison of speed of original deterministic algorithms and randomized versions on test-bed problems.

complete algorithms is thus explained by the variance one would expect in any one run of a randomized algorithm. Furthermore, by analyzing the shape of the cost distribution we developed simple techniques that provably *reduce* the mean solution time.

For our experiments, we used known hard problem instances from scheduling, planning, and circuit synthesis, and a state of-the-art satisfiability engine (Satz, by Li and Anbulagan (1997)), and a highly efficient CSP solver built using the ILOG C++ constraint programming library (Puget and Leconte 1995). It is important to note that in both cases the underlying deterministic complete search engines are among the fastest (and on many problems, *the* fastest) in their class. Thus, the techniques discussed in this paper extend the range of complete methods to problems that were previously beyond their reach. For a preview of our main results, see Table 1. The table shows how our randomization strategy enabled us to solve several previously unsolved problem instances, and other instances were solved up to an order of magnitude faster. Given the techniques' simplicity and generality, our approach can be easily adapted to improve the performance of other backtrack-style search methods used in planning, scheduling, and other tasks of interest to AI.

Problem Domains

Our problem domains are timetable scheduling, planning, and circuit synthesis. The first is formalized as a CSP problem, and the latter two as propositional satisfiability.

Timetabling consists in determining whether there exists a feasible schedule that takes into consideration a set of pairing and distribution constraints. More specifically, we consider problems derived from sports scheduling applications. The literature in this area is growing, and one can begin to get a sense of the range and mathematical difficulty of the problems encountered (McAloon *et al.* 1997; Nemhauser and Trick 1997; and Schreuder 1992). Here we consider the timetabling problem for the classic "round-robin" schedule: every team must play every other team exactly once. The problem is formally defined as follows:

1. There are N teams (N even) and every two teams play each other exactly once.
2. The season lasts $N-1$ weeks.
3. Every team plays one game in each week of the season.
4. There are $N/2$ periods and, each week, every period is scheduled for one game.
5. No team plays more than twice in the same period over the course of the season.

Up to 8-team problems are relatively simple and can be done by brute force. However, the combinatorics of this scheduling problem are explosive. For an N team league, there are $N/2 \cdot (N-1)$ matchups (i,j) with $0 \leq i < j < N$ to be played. A schedule can be thought of as a permutation of these matchups. So, for N teams the search space size is $(N/2 \cdot (N-1))!$, *i.e.*, the search space size grows as the factorial of the square of $N/2$. Published algorithms for this problem all scale poorly, and the times for our *deterministic* solver (as shown in Table 1) are among the best (see also Gomes *et al.* 1998b).

The second domain is planning. Kautz and Selman (1996) showed that propositional SAT encodings of difficult STRIPS-style planning problems could be efficiently solved by SAT engines. While both a complete backtrack-style engine and an incomplete local-search engine worked well on moderate-sized problems, the largest problems from the domain of logistics scheduling could only be solved by local search. However, it turns out that the deterministic version of Satz can solve all of the logistics instances from that paper in less than 2 minutes. Therefore we constructed a still-larger planning problem, labeled "logistics.d". This domain involves moving packages on trucks and airplanes between different locations in different cities. While the largest logistics problem from the Kautz and Selman (1996) paper involved 1,141 variables, "logistics.d" involves 2,160 variables.

The final domain is circuit synthesis. Kamath *et al.* (1993) developed a technique for expressing the problem of synthesizing a programmable logic array (PLA) as a propositional satisfiable problem. The statement of the problem includes a table specifying the function to be computed, and an upper-bound on the number of gates that may appear in the circuit. In general, these problems become more difficult to solve as the number of gates is reduced, until the limit is reached where the instance is unsatisfiable. These problems are quite hard to solve with complete SAT procedures, and have been used as part of the test-beds for numerous SAT competitions and research studies. The problems considered in this paper, "3bit-adder-32" and "3bit-adder-31" are (as one would guess) based on synthesizing a 3-bit adder using 32 and 31 gates respectively. Although Selman and Kautz (1993) solve the instances using local search, no one has previously solved either using a backtrack-style procedure.

Randomizing Complete Search Engines

We now consider general techniques for adding randomization to complete, systematic, backtrack-style search procedures. Such a procedure constructs a solution incrementally. At each step a heuristic is used to select an operation to be applied to a partial solution, such as assigning a value to an unassigned variable. Eventually either a complete solution is found, or the algorithm determines that the current partial solution is inconsistent. In the latter case, the algorithm backtracks to an earlier point in its search tree.

If several choices are heuristically determined to be equally good, then a deterministic algorithm applies some fixed rule to pick one of the operations; for example, to select the lowest-numbered variable to assign. The most obvious place to apply randomization, therefore, is in this step of tie-breaking: if several choices are ranked equally, choose among them at random. Even this simple modification can dramatically change the behavior of a search algorithm, as we will see in the section on CSP below.

However, if the heuristic function is particular powerful, it may rarely assign more than one choice the highest score. To handle this, we can introduce a "heuristic equivalence" parameter to the algorithm. Setting the parameter to a value H greater than zero means all choices who receive scores within H-percent of the highest score are considered equally good. This expands the choice-set for random tie-breaking.

With these changes each run of the search algorithm on a particular instance will differ in the order in which choices are made and potentially in time to solution. As we will discuss in detail below, it can be advantageous to terminate searches which appear to be "stuck", exploring a part of the space far from a solution. Therefore we will also introduce a "cutoff" parameter, that limits search to a specified number of backtracks. When the cutoff is reached, the algorithm is restarted at the root of the search tree.

We should note that introducing randomness in the branching variable selection does not effect the completeness of the backtrack-style search. Some basic bookkeeping (only linear space) ensures that the procedures do not revisit any previously explored part of the search space, which means that we can still determine inconsistencies, unlike local search methods. The cutoff parameter does limit the size of the space that can be searched exhaustively between restarts. In practice, we gradually increase the cutoff, to allow us to determine inconsistencies, if necessary.

A variable-order randomization and restart strategy was employed in Crawford and Baker's (1994) "probing" algorithm for SAT. Despite the fact that it performed no backtracking at all, it was shown to solve a number of examples. Even though, the "power of randomization" in combinatorial search has been informally recognized by others (for recent work in scheduling domains, see e.g., Bresina 1996 and Oddi and Smith 1997), our work provides the first explanation for the potential success of this kind of strategy, in terms of heavy-tailed distributions (Gomes *et al.* 1998a). As we will see, our data also shows that there is often a clear optimal cutoff value; simply probing down with unit propagation but no backtracking can be ineffective. For example, in Table 3 we have a 0% success rate for a cutoff value of 2. More recently, Bayardo and Schrag (1997) introduced a backtrack-style solver, rel-sat, that included randomized tie-breaking and restarts, but with only a fixed, high cutoff value. The focus of that work was on the backtracking technique, rather than the effect of restarts.

The first complete search algorithm we randomized was a CSP solver. ILOG SOLVER is a powerful C++ constraint programming library (Puget and Leconte 1995). For the round-robin scheduling problems discussed below, we used the library to build a deterministic, backtrack-style CSP engine. (See Dechter (1991) and Freuder and Mackworth (1994) for an overview of basic CSP algorithms.) It employs the first-fail heuristic for variable assignment, which selects the variables with the smallest domain first; ties are broken lexicographically. The performance of this deterministic version already matches or exceeds all the published results on solving these types of problems. We then randomized the solver by breaking ties randomly, and adding a cutoff parameter (Gomes *et al.* 1998b).

The second algorithm we randomized was for propositional satisfiability. One of the fastest complete search engines for propositional satisfiability testing is the Satz system of Li and Anbulagan (1997). Satz is a version of the Davis-Putnam-Loveland procedure (Davis *et al.* 1962), with a heuristic based on choosing a branch variable that maximizes a function of the number of the unit propagations performed when it is set positively or negatively. Satz is the fastest deterministic SAT procedure we have found for the instances discussed in this paper. It can often solve smaller instances of these types with less than 100 backtracks. Because its heuristic usually chooses a single branching variable without ties, we added a heuristic equivalence parameter to enlarge the choice-set.

Heavy-Tailed Cost Distributions

In previous work (Gomes *et al.* 1998a), we show that the tail behavior of randomized complete backtrack style methods is often best modeled using distributions which asymptotically have tails of the Pareto-Lévy form, *viz.*

$$\Pr\{X > x\} \sim C.x^{-\alpha}, \quad x > 0 \qquad (1)$$

where $\alpha > 0$ is a constant (Mandelbrot 1960; and Samorodnitsky 1994). These are heavy-tailed distributions, *i.e.*, distributions whose tails have a *power law decay*. The constant α is called the *index of stability* of the distribution. For $\alpha < 2$, moments of X of order less than α are finite while all higher order moments are infinite, *i.e.*, $\alpha = \sup\{a > 0 : \mathrm{E}|X|^a < \infty\}$. For example, when $\alpha = 1.5$, the distribution has a finite mean but no finite variance. With $\alpha = 0.6$, the distribution has neither a finite mean nor a finite variance.

If a Pareto-Lévy tail is observed, then the rate of decrease of the distribution is a power law. (Standard distributions exhibit exponential decay.) From (1), we have $1 - F(x) = \Pr\{X > x\} \sim C.x^{-\alpha}$, so the complement-to-one of the cumulative distribution, $F(x)$, also decays according to a power law. Given the power law decay of the complement-to-one of the cumulative distribution of a heavy-tailed random variable, its log-log plot should show an approximately linear decrease in the tail. Moreover, the slope of the observed linear decrease provides an estimate of the index α.

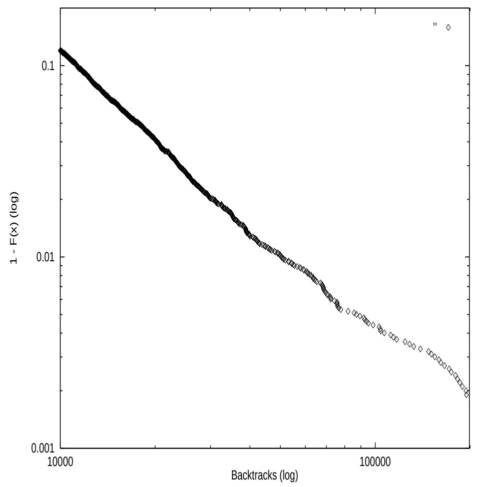

Figure 1: Log-log plot of the tail of 12 team round-robin scheduling.

Figure 1 shows the log-log plot of the tail ($X > 10,000$) of the complement-to-one of the cumulative distribution, 1-F(x), for our 12 team round-robin problem. The linear nature of the tail in this plot directly reveals heavy-tails of the Pareto-Lévy type.

To complement our visual check of heavy-tailed behavior of Figure 1, we calculate the maximum likelihood estimate of the index of stability (the value of α): For our round-robin scheduling problem, for $N = 12$, we obtain $\alpha = 0.7$, which is consistent with the hypothesis of infinite mean and infinite variance, since $\alpha \leq 1$.[1]

So far, we have identified heavy-tailed behavior of the cost distribution to the right of the median. The heavy tail nature shows that there is a computationally significant fraction of very long runs, decaying only at a polynomial rate. The strategy of running the search procedure with a cutoff near the median value of the distribution clearly avoids these long runs in the tail.

However, our experiments in Gomes (1998a) also suggest a heavy tail phenomenon on the left-hand side of the median value of the cost distribution, which means that the success rate for a solution only increases polynomially with the number of backtracks. This explains how a relatively low cutoff value still gives a sufficiently high success rate to allow us to solve a problem instance. For example, for our round-robin scheduling problems with $N = 16$, we observed several runs that took less than 200 backtracks, compared to a median value of around 2,000,000. For $N = 18$, we ran with a cutoff of $500,000$ and solved the instance after 20 tries. Each try took about 1 hour, and the successful run took 350,632 backtracks.

Tails on the left are also characterized by an index of stability. Based on our data (Gomes 1998a), we conjecture that α for the tail on the left is less than 1.0 on hard combinatorial search problems. This conjecture has strong implications in terms of algorithm design: It means that in order to obtain the minimal expected run time, a preferred strategy consists of relatively short runs of a randomized backtrack-style procedure.

We do not wish to give the impression that *every* search problem gives rise to a heavy-tailed distribution. In fact, doing so would give rise to the suspicion that the distributions we found were an artifact of our methodology, rather than a real phenomena of the problem domain! One domain in which we have *not* found heavy-tails is on blocks-world planning problems. The hardest blocks-world problem from Kautz and Selman (1996) is blocks-world.d, and it can be solved by deterministic Satz in 30 minutes. We ran the randomized version of Satz on this instance at a wide range of cutoff values and heuristic equivalence settings.

[1]Of course, the computational cost of complete backtrack-style algorithms has a finite upper-bound. However, since we are dealing with NP-complete problems, this upper-bound is exponential in the size of the problem, which means that *de facto*, for realistic-size hard instances, it can be treated as infinite for practical purposes: no practical procedure can explore the search full space.

The optimal equivalence parameter setting was 30%. However, over a range of cutoff values, there was no evidence of a heavy-tailed distribution, and, therefore, randomization only slightly increases the effectiveness of Satz: the mean cost is 23 minutes. Further studies are needed to determine exactly what characteristics of combinatorial search problems lead to heavy-tailed behavior.

Boosting Performance by Randomization and Restarts

So far, we have discussed how heavy-tailed probability distributions underlie the large variability observed when running a randomized backtrack-style procedure on a variety of problem instances. We can obtain more efficient and more predictable procedures by running the search up to a certain cutoff point and then restarting at the root of the tree. Restarts clearly prevent the procedure from getting trapped in the long tails on the right of the distribution. In addition, a very low cutoff value can also be used to exploit the heavy-tails to the left of the median, and will allow us to solve previously unsolved problem instances after a sufficient number of restarts. In Table 1, the mean solution times in the "Randomized" column are based on empirically determined near-optimal cutoff values. For each randomized solution time the standard deviation is of the same order of magnitude as the mean. This is to be expected because the distribution is geometric, as will be shown in the next section. Without restarts, of course, the variance and mean tend to infinity to a first approximation.

We will now discuss these results in more detail.

Our deterministic CSP procedure on the round-robin scheduling problem gives us us a solution for $N = 14$ in about 411 seconds. (Experiments ran on a 200MHz SGI Challenge.) We could not find a solution for $N = 16$ and $N = 18$. Apparently, the problem quickly becomes very difficult, even for moderate values of N. The subtle interaction between global and local constraints makes the search for a globally consistent solution surprisingly hard.

For problems for which we can empirically determine the overall cost profile, we can calculate an *optimal* cutoff value to minimize the expected cost of finding a solution. Our main interest, however, is in solving previously unsolved instances, such as the $N = 16$ and $N = 18$ case. These problems are too hard to obtain a full cost distribution. For example, for $N = 16$, running with a cutoff of 1,000,000 gives a success rate of less than 40%, so we do not even reach the median point of the distribution. Each run takes about 2 hours to complete. (We estimate that the median value is around 2,000,000. Our deterministic procedure apparently results in a run that still lies to the right of the expected median cost.) In order to find a good cutoff value for very hard problem instances, the best available strategy is a trial-and-error process, where one experiments with various cutoff values, starting at relatively low values, since the optimal cutoff for these problems tends to lie below the median value of the distribution. This can be seen from Table 2, which gives the expected cost (backtracks) for finding a solution for $N = 16$ for a range of cutoff values. The optimal cutoff is around 5.10^4, resulting in an expected cost per solution of 7.10^5 backtracks ($\approx$ 1.4 hrs). For the $N = 18$ case, we ran with a cutoff of 5.10^5, and found a solution after approximately 22 hours.[2]

cutoff	succ. rate	mean cost ($\times 10^6$)
200	0.0001	2.2
5,000	0.003	1.5
10,000	0.009	1.1
50,000	0.07	0.7
100,000	0.06	1.6
250,000	0.21	1.2
1,000,000	0.39	2.5

Table 2: Solving the 16-team robin-robin scheduling problem for a range of cutoff values.

cutoff	succ. rate	mean cost
2	0.0	>300,000
4	0.00003	147,816
8	0.0016	5,509
16	0.009	1,861
32	0.014	2,405
250	0.018	13,456
16000	0.14	107,611
128000	0.32	307,550

Table 3: Solving the logistics.d problem for a range of cutoff values.

Table 3 gives the performance of Satz for a range of cutoff values on the logistics.d instance. Again, there is a clear optimal value: In this case, it's surprisingly low, 16 backtracks. Despite the low success rate (less than 1%) at this cutoff value, the overall performance is close to optimal here, requiring around 1,800 backtracks total per solution, which takes around 95 seconds. Compare this with the 108 minutes for the deterministic version of Satz. It's important to note that the 108 minutes run is not just an "unlucky" determinist run. Given the shape of the underlying heavy-tailed distribution, most runs take more than 100,000 backtracks (over 1 hour). The trick is to exploit the fact that we

[2]Since the submission of this paper, a lot of progress has been made in terms of solving larger instances (McAloon *et al.* in preparation). By using multiple threads on a 14 processor Sun system, 26 and 28 teams schedules were generated, which is the record as of this writing (Wetzel and Zabatta, 1998). We believe these numbers can be improved upon with our randomization technique.

have a non-negligible probably of solving the instance in a *very* short run. Our fast restart strategy exploits this.

See Table 1 for other improvements due to randomization. Until now, the 3bit-adder problems had not been solved by any backtrack-style procedure. On the blockworld problem, we obtain little improvement, which can be attributed to the absence of heavy-tails as discussed above.

These results show that introducing a stochastic element into a backtrack-style search procedure, combined with an appropriate restart strategy, can significantly enhance the procedure's performance. In fact, as we see here, it allows us to solve previously unsolved problem instances.

A Formal Analysis of Restarts

In this section we formalize the strategy of restarts S of a complete stochastic procedure A. We derive the probability distribution of S assuming the full knowledge of the probability distribution of A. We demonstrate that the probability distribution associated with S does not exhibit heavy tails. Furthermore, S has a finite mean and variance, even if the stochastic procedure A has an infinite mean and variance.

Let us consider a complete stochastic procedure and associate with it the random variable A, where A is the number of backtracks that it takes to find a solution or prove that it does not exist. Let us now consider the following stochastic strategy for running A: run A for a fixed number of backtracks c (the cutoff); if A finds a solution or proves it does not exist, then our stochastic strategy has also found a solution (or proved that it does not exist) and it stops. Otherwise, restart A from the beginning, using an independent random seed, for another c backtracks, and so on. Define S as the number of backtracks that the stochastic strategy of restarts of A with cutoff c takes to find a solution or prove that it does not exist. Let's assume that we know P[A$\leq c$], *i.e.*, the probability that the stochastic procedure A will find a solution or prove that it does not exist in no more than c backtracks. The sequence of runs of A executed by our restart strategy are independent, and therefore they can be seen as a sequence of Bernoulli trials, in which the success consists in finding a solution (or proving that it doesn't exist) before the end of the run.

It's convenient to also define a random variable R, giving the number of restarts until a solution is found (or the instance is shown inconsistent). Note that $R = \lceil S/c \rceil$. R follows a *geometric distribution* with parameter $p = P[A \leq c]$. The probability of the tail of S, $P[S > s]$, is given by

$$P[S > s] = (1-p)^{\lfloor s/c \rfloor} \; P[A > s \; mod \; c]$$

Taking into consideration that $R = \lceil S/c \rceil$ and that it follows a geometric distribution (exponential decay; finite mean and variance), it follows that the tail of the distribution of S also exhibits exponential decay and S has a finite mean and variance.

We should emphasize that when adopting a low cutoff the strategy of restarts partially eliminates the heavy tail on the left: the lower the cutoff, the shorter the tail. This is true since the distribution of S exhibits exponential decay for $S >$cutoff.

Conclusions

Building on our previous work on heavy-tailed behavior in combinatorial search (Gomes et al. 1998a), we have shown that performance of complete, backtrack-style search algorithms on hard real-world problems can be greatly enhanced by the addition of randomization combined with a rapid restart strategy. Speedups of several orders of magnitude were observed, and some test problem instances were solved for the first time by any backtrack-style procedure.

The success of our approach is based on exploiting the heavy-tailed nature of the cost distributions. We saw that in most of the domains we found that "outliers" on *both* sides of the median occur with a relatively high frequency. Heavy-tails to the right of the median cause the mean solution time to grow without bounds. Adding cutoffs and restarts to the search algorithm, however, both theoretically and empirically eliminate the heavy-tail and bound the mean. Heavy-tails to the left of the mean can be exploited by performing many rapid restarts with short runs, leading to a further dramatic decrease in expected solution time.

We applied the randomization techniques to two state-of-the-art search engines for CSP and propositional satisfiability. We were able to solve hard round-robin scheduling instances of up to size 18, when the corresponding deterministic version could only handle instances up to size 14. In the domain of planning as satisfiability, we extended the range of logistics problems that could be solved by complete methods from problems containing 1,141 variables to ones involving 2,160 variables (solved with mean cost of 95 seconds).

It would be interesting to explore our randomization approach in context of other backtrack-style approaches, such as dynamic backtracking (Ginsberg 1993). We believe that the generality of the approach will lead to further advances in planning, scheduling, diagnosis, game-playing, and other areas of AI.

Acknowledgments The first author is sponsored by the Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-98-1-0008 and the Air Force Office of Scientific Research, under the New World Vistas Initiative, AFOSR NWV project 2304, LIRL 97RL005N25. The second author is supported by an NSF Faculty Early Career Development Award. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

References

Alt, H., Guibas, L., Mehlhorn, K., Karp, R., and Wigderson A. (1996). A method for obtaining randomized algorithms with small tail probabilities. *Algorithmica*, 16, 1996, 543–547.

Bayardo, Roberto J., and Schrag, Robert C. (1997). Using CSP look-back techniques to solve real-world SAT instances. *Proc. AAAI-97*, New Providence, RI, 1997, 203–208.

Bresina, J. (1996) Heuristic-biased stochastic sampling. *Proc. AAAI-96*, Portland, OR, 1996.

Crawford, J. M., and Baker, A. B. (1994). Experimental results on the application of satisfiability algorithms to scheduling problems. *Proc. AAAI-94*, Seattle, WA, 1092–1097.

Davis, M., Logemann, G., and Loveland, D. (1962). A machine program for theorem proving. *Comm. ACM*, 5, 1962, 394–397.

Dechter, R. (1991). Constraint networks. *Encyclopedia of Artificial Intelligence* John Wiley, New York (1991) 276-285.

Freuder, E. and Mackworth, A., eds. (1994). *Constraint-based reasoning.* MIT Press, Cambridge, MA.

Gent, Ian P. and Walsh, Toby (1994). Easy problems are sometimes hard. *Artificial Intelligence*, (70)1-2, 335–345.

Ginsberg, M. (1993). Dynamic Backtracking. *Journal of Artificial Intelligence*, Vol. 1, 25–46.

Gomes, C.P. and Selman, B. (1997). Problem structure in the presence of perturbations. *Proc. AAAI-97*, New Providence, RI, 221–226.

Gomes, C.P., Selman, B.,and Crato, N. (1998a). Heavy-Tailed Phenomena in Combinatorial Search, 1998. (submitted for publication)

Gomes, C.P. and Selman, B., McAloon, K., and Tretkoff C. (1998b). Randomization in Backtrack Search: Exploiting Heavy-Tailed Profiles for Solving Hard Scheduling Problems. To appear in: *Proc. AIPS-98.*

Kamath, A.P., Karmarkar, N.K., Ramakrishnan, K.G., and Resende, M.G.C. (1993). An Interior Point Approach to Boolean Vector Function Synthesis. *Proc. 36th MSCAS*, 185–189.

Kautz, H. and Selman, B. (1996). Pushing the envelope: planning, propositional logic, and stochastic search. *Proc. AAAI-1996*, Portland, OR.

Li, Chu Min and Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. *Proc. IJCAI-97*, Kyoto, Japan, 1997.

Luby, M., Sinclair A., and Zuckerman, D.
(1993). Optimal speedup of Las Vegas algorithms. *Information Process. Lett.*, 17, 1993, 173–180.

Mandelbrot, Benoit, B. (1960). The Pareto-Lévy law and the distribution of income. *International Economic Review* 1, 79–106.

McAloon, K., Regin, J-C., Tretkoff C. and Wetzel G. (1998). Constraint-Based Programming for Sports League Scheduling. Manuscript in preparation 1998.

McAloon, K., Tretkoff C. and Wetzel G. (1997). Sports League Scheduling. *Proceedings of Third Ilog International Users Meeting*, 1997.

Nemhauser, G., and Trick, M. (1997). Scheduling a major college basketball conference. Georgia Tech., Technical Report, 1997.

Oddi A. and Smith, S. (1997) Stochastic procedures for generating feasible schedules. *Proc. AAAI-97*, New Providence, RI, 1997.

Puget, J-F., and Leconte, M. (1995). Beyond the Black Box: Constraints as objects. *Proceedings of ILPS'95*, MIT Press, 513–527.

Samorodnitsky, Gennady and Taqqu, Murad S. (1994). *Stable Non-Gaussian Random Processes: Stochastic Models with Infinite Variance*, Chapman and Hall, New York.

Schreuder, J. A. M. (1992). Combinatorial Aspects of Construction of Competition Dutch Professional Football Leagues, *Discrete Applied Mathematics* 35 (1992) 301-312.

Selman, B. and Kautz, H. (1993). Domain-independent extensions to GSAT: solving large structured satisfiability problems. *Proc. IJCAI-93*, Chambéry, France, 290–295.

Smith, B. and Grant S.A. (1996). Sparse constraint graphs and exceptionally hard problems. *IJCAI-95*, 646–651, 1995. Full version in AIJ (Hogg et al. 1996).

Wetzel, G. and Zabatta, F. (1998). CUNY Graduate Center CS Technical Report, 1998.