Chapter 1

# RANDOMIZED BACKTRACK SEARCH

## *Extending the reach of complete search methods*

Carla P. Gomes
*Dept. of Computer Science*
*Cornell University*
*Ithaca, NY 14853*
gomes@cs.cornell.edu

**Abstract**

Randomized search methods have greatly extended our ability to solve hard computational problems. In general, however, we think of randomization in the context of local search. While local search methods have proved to be very powerful, in some situations they cannot supplant complete or exact methods due to their inherent limitation: Local search methods cannot prove inconsistency or optimality.

In recent years, we have seen the emergence of an active research area focusing on the study and design of complete randomized search procedures. In this chapter we introduce the ideas and concepts underlying such randomized procedures. In particular, we describe a new generation of backtrack-style methods that exploit randomization while guaranteeing completeness. We analyze the run time distributions of such methods and show that quite often they exhibit so-called heavy tails. *Heavy tailed* distributions are non-standard distributions that have infinite moments (*e.g.*, an infinite mean or variance). Such non-standard distributions have been observed in areas as diverse as economics, statistical physics, and geophysics. The understanding of the heavy-tailedness of complete backtrack search methods has led to the design of more efficient procedures. We describe such new techniques. In particular, we present different ways of randomizing complete backtrack style methods and show how restart and portfolio strategies can increase performance and robustness of such methods. We also provide formal models of heavy-tailed behavior in combinatorial search and results on performance guarantees for restart strategies.

**Keywords:** Complete backtrack search, randomization, heavy tails, restarts, portfolio strategies

## 1. Introduction

One of the most surprising discoveries in the area of algorithm design is that the incorporation of an element of randomness into the computational process can lead to a significant speedup over purely deterministic methods. For example, randomness lies at the core of highly effective methods for determining whether a given number is prime— a key problem in modern cryptography ( Agrawal et al., 2002; Goldwasser and Kilian, 1986; Rabin, 1980; Solovay and Strassen, 1977)— and it is a key component in the popular simulated annealing method for solving optimization problems (Kirkpatrick et al., 1983). More generally, randomization and stochastic strategies have been very successful in local search. Local search methods are often used to solve large-scale combinatorial and optimization problems that are beyond the reach of complete search methods. Such methods start with an initial solution, not necessarily feasible, and improve upon it by performing small "local" changes. There are several ways of implementing local search methods, depending on the choice of initial solution, the types of "local" changes allowed, and feasibility and cost of (intermediate) solutions. In chapter X, Lodi *et al.*, provide a detailed overview of local search.

Two of the key features of local search methods are their flexibility and their ability to sample large portions of the search space. However, local search methods are inherently incomplete, that is, they cannot establish inconsistency or prove optimality, which is critical in many real-world applications.

Given the inherent intractability of NP-complete problems, one often has to resort to exhaustive search procedures that explore the large space of potential solutions in order to find an actual solution (or prove that no solution exists); in the case of optimization problems, the goal is to prove the optimality of a solution. These methods are generally implemented as "backtrack-style" search procedures. The idea is to iteratively construct a solution. If it becomes apparent that the current solution path is neither feasible, nor optimal, the procedure backtracks to explore another solution path. This general strategy underlies complete search methods for solving large classes of combinatorial problems, such as Boolean satisfiability (SAT), constraint programming (CP), and integer programming (IP).

The performance of backtrack-style search methods can vary dramatically, depending on the way one selects the next variable to branch on (the "variable-selection heuristic") and in what order the possible values are assigned to a variable (the "value selection heuristic"). Branching heuristics play a central role in guiding backtrack search procedures toward regions of the search space that contain solutions. In some applications, in fact, backtrack search methods perform better than local search methods, largely as a result of a combination of highly tuned search heuristics and strong propagation mechanisms.

Unfortunately, quite often the branching heuristics provide incorrect search guidance, forcing the procedure to explore large subtrees of the search space that do not contain any solution. As a consequence, backtrack search methods exhibit a large variation in performance. For example, we see significant differences on runs of different heuristics, runs on different problem instances, and, for randomized backtrack search methods, runs with different random seeds. The inherently exponential nature of the search process appears to magnify the unpredictability of search procedures. In fact, it is not uncommon to observe a backtrack search procedure "hang" on a given instance, whereas a different heuristic — or even just another randomized run — may solve that instance quickly.

Various researchers studying the computational nature of search methods on combinatorial problems have informally observed this high variance in the performance of backtrack search methods. A related phenomenon is that of so-called exceptionally hard instances. An instance is considered to be *exceptionally hard*, for a particular search algorithm when it lies in the region where almost all problem instances are satisfiable (*i.e.*, the under-constrained area) but, for a given algorithm, is considerably harder to solve than other similar instances, and even harder than most of the instances in the critically constrained area (Gent and Walsh, 1993; Smith and Grant, 1995). Selman and Kirkpatrick, 1996, observed that, by simply renaming the variables, one could solve exceptionally hard instances easily. Thus, the "hardness" does not necessarily reside in the instances themselves, but rather in the instances in combination the details of the deterministic algorithm.

The extreme variance in performance of search algorithms has led researchers studying the nature of computationally hard problems to use the median, rather than the mean, to characterize search difficulty, since the behavior of the mean tends to be quite *erratic*, while the median is generally much more stable ( Cook and Mitchell, 1998; Gent and Walsh, 1993; Hogg et al., 1996; Kirkpatrick and Selman, 1994; Mitchell and Levesque, 1996; Vandengriend and Culberson, 1999). More recently, it has been shown that the full runtime distributions of search methods provide a better characterization of search methods than just the moments and the median and that they yield much useful information for the design of algorithms (Frost et al., 1997; Gomes et al., 1997; Gomes and Selman, 1997a; Hoos, 1998; Kwan, 1995; Rish and Frost, 1997). In particular, Gomes et al., 1997, showed that the runtime distributions of complete backtrack search methods quite often exhibit *heavy-tails*. Heavy-tailed distributions are characterized by infinite moments (*e.g.*, they can have infinite mean, or infinite variance). Heavy tails have been observed in aggregated runtime distributions of backtrack search methods when considering a collection of instances of the same class (*e.g.*, random binary CSP instances generated with the same parameter space). They have also been encountered when running a randomized backtrack

search procedure on the *same* instance several times, but where *randomization was only used to break ties in the variable and/or value selection*. Gomes *et al.* further showed how randomized *restarts* of search procedures can dramatically reduce the variance in the search behavior. In fact, they demonstrated that a search strategy with restarts provably eliminates heavy tails (Gomes et al., 2000). Consequently, the introduction of an element of randomness into a backtrack search procedure can lead to a family of runs with different runtimes, which can be exploited in conjunction with restart strategies in order to obtain a procedure with considerably better performance, *on average*.

The stringent completeness requirements of several applications has led to a tremendous growth in research aimed at boosting performance of complete backtrack-style methods. A good example is the area of software and hardware verification. Verification problems can be encoded as Boolean satisfiability (SAT) problems. The SAT community has embraced and extended randomization and restart techniques to boost performance of complete search methods for SAT. In fact, randomization and restart strategies are now an integral part of many state-of-the-art SAT solvers (Bayardo and Schrag, 1997; Goldberg and Novikov, 2002; Li, 1999; Lynce and Marques-Silva, 2002a; Moskewicz et al., 2001; Zhang, 2002). The world's fastest SAT solver at the present time, called Chaff, incorporates such techniques (Chaff01). In Chaff, restarting is combined with clause learning, another technique central to Chaff's overall effectiveness. In clause learning, whenever a conflict (*dead end*) is reached, a *nogood* is recorded and carried over between restarts (Bayardo and Schrag, 1997; Marques-Silva and Sakallah, 1999; Lynce and Marques-Silva, 2002c). Informally, a nogood provides information about inconsistent (partial) assignments inferred from the search performed thus far. Chaff can handle problem instances with over one million variables and four million constraints.

In summary, the study and design of complete randomized search procedures is an emerging and active research area. The understanding of the heavy-tailed nature of the distributions underlying such search methods has led to the design of more efficient backtrack search techniques. In this chapter we describe such new techniques. In section 2 we discuss different ways of randomizing complete backtrack-style methods. In section 3 we present formal models that provably exhibit heavy-tailed behavior. In section 4 we characterize in detail *heavy*- and *fat*-tailedness. In section 5 we provide empirical distributions of randomized backtrack search methods that illustrate heavy-tailed behavior. In sections 6 and 7 we show how restart and portfolio strategies can increase performance and robustness of randomized backtrack search methods. Our conclusions, together with suggested directions for future research, are presented in section 8.

## 2. Randomization of Backtrack Search Methods

An emerging area of research is the study of Las Vegas algorithms. These are *exact* randomized algorithms; they always return a model satisfying the constraints of the search problem, proving optimality in the case of optimization problems— or proving that no such model exists in the case of infeasible problems ( Motwani and Raghavan, 1995). Contrast Las Vegas algorithms with, for example, Monte Carlo algorithms, randomized methods that can provide incorrect answers. The class of randomized local search methods is an example of a class of Monte Carlo algorithms, given their inherent incompleteness.

A randomized algorithm can be viewed as a probability distribution on a set of deterministic algorithms. The behavior of a randomized algorithm can vary even on a single input, depending on the random choices made by the algorithm. The classical adversary argument for establishing lower bounds on the runtime of a deterministic algorithm is based on the construction of a input on which the algorithm performs poorly. Therefore, different inputs may have to be constructed for each deterministic algorithm. When we consider a randomized algorithm, we are implicitly considering a choice of one algorithm at random from a whole family of algorithms, and while an adversary may be able to construct an input that foils one (or a small fraction) of the deterministic algorithms in the set, it is more difficult to devise inputs that are likely to defeat a randomly chosen algorithm. For many problems, randomized algorithms have been shown to be more efficient than the best-known deterministic algorithms, and, in general, they are simpler to describe than deterministic algorithms of comparable performance (Motwani and Raghavan, 1995).

We consider general techniques for randomizing backtrack search algorithms while maintaining completeness. Backtrack search provides a general framework for solving hard computational problems.

Backtrack search methods construct 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 (or suboptimal), in which case the algorithm backtracks to an earlier point in its search tree.

Figure 1.1 illustrates the application of a simple backtrack search algorithm to the Boolean satisfiability problem, looking for an assignment to the variables $A$, $B$, and $C$ that satisfies the formula $(A \vee \neg B \vee \neg C) \wedge (B \vee \neg C) \wedge (A \vee C)$. The figure shows two different executions of the backtrack search procedure. In the search tree shown on the left, the variable-choice heuristic picks the variables in the order $A$, $B$, then $C$, and always tries the value 0 before the value 1. Execution is traced along the left-most branch of the tree until a constraint is determined to be false, creating a dead-end (marked X). The algorithm then

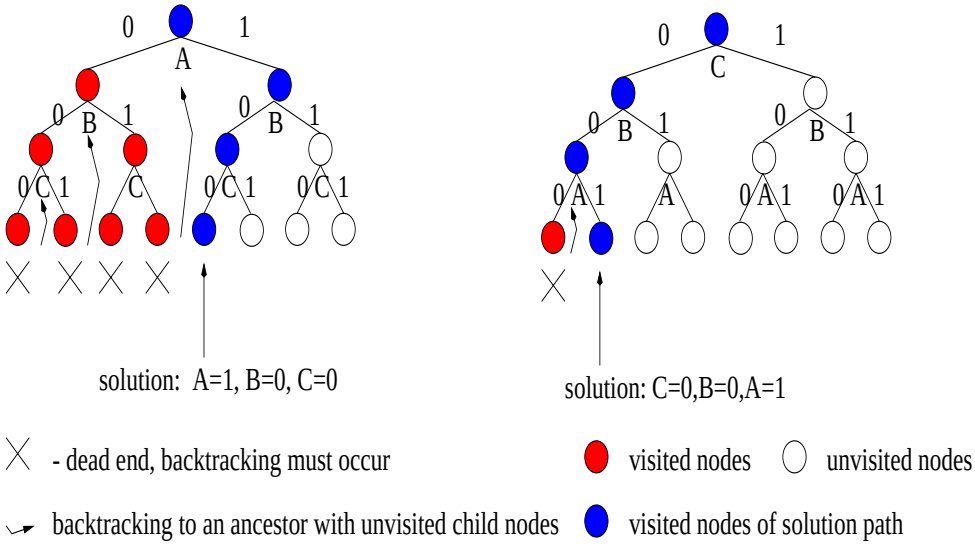



*Figure 1.1.* Two different executions of a backtrack search methods looking for an assignment to the variables A, B, C that satisfies the formula $(A \vee \neg B \vee \neg C) \wedge (B \vee \neg C) \wedge (A \vee C)$

backtracks up a level in the tree and tries the right branch (assigning 1 to the variable). To backtrack from a right branch, the algorithm goes up two levels, and so on. In the search tree shown on the right, the variable-choice heuristic picks the variables in the order $C$, $B$, then $A$, and, again, always tries the value 0 before the value 1. The execution depicted on the right is shorter than the one on the left.

There are several techniques to enhance the performance of backtrack-style search methods, in particular *look-ahead* and *look-back* techniques. Look-ahead techniques exploit information about the remaining search space. Pruning, cutting planes, and constraint-propagation mechanisms that enforce consistency on the basis of previous assignments are examples of look-ahead techniques. Look-back techniques, on the other hand, include *learning* and forms of *intelligent backtracking*, *i.e*, techniques to reason about the search space already explored. For example, in intelligent backtracking, the causes of inconsistency are analyzed at the time at which backtracking is to occur, so that branches that have been proved not to contain solutions can be skipped.

There are, therefore, several opportunities to introduce randomization into a backtrack search method, basically at the different decision points where the next operation to be performed by the backtrack solver is to be selected. In particular, randomization can be incorporated into the following operations:

- variable selection

- value selection
- look-ahead procedures
- look-back procedures
- backtrack-points
- target-point selection (at the time of backtracking)

In general, backtrack search procedures use *deterministic* heuristics to select the next operation. Highly tuned heuristics are used for variable and value selection. 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, using lexicographical order. Therefore, the most straightforward way to apply randomization is in this tie-breaking step: If several choices are ranked equally, choose among them at random (Gomes et al., 1997; Gomes et al., 2000). Even this simple modification can dramatically change the behavior of a search algorithm, as we will see below. If the heuristic function is particularly powerful, however, it may rarely assign the highest score to more than one choice. 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 that receive scores within $H$ percent of the highest score are considered equally good. This expands the choice set for random tie-breaking (Gomes et al., 1998a).

A more general procedure, which includes the tie-breaking and the $H$-percent equivalence-window procedures as special cases, imposes a probability distribution on the set of possible values for a given operation. For example, in the tie-breaking case, we are implicitly assigning a uniform probability to the values that are considered by the heuristic to be equally good, while in the case of using the $H$-percent window, we assign a uniform probability to all choices whose scores lie within $H$ percent of the highest score. An alternative would be to impose a different distribution on the values within the $H$-percent window (for example, an exponential distribution), biasing the selection toward the highest score.

Random reordering is another way of "randomizing" an algorithm. This technique involves randomly reordering the input data, followed by the application of a deterministic algorithm (Motwani and Raghavan, 1995). Walsh, 1999, applied this technique to study the runtime distributions of graph-coloring problems, using a deterministic, exact coloring algorithm based on DSATUR ( Trick, 1996; see also section 5.4).

A key issue when introducing randomization into a backtrack search algorithm is that of guaranteeing completeness. We note that the introduction of randomness into the branching-variable/value selection does not affect the

completeness of the backtrack search. Some basic bookkeeping ensures that the procedures do not revisit any previously explored part of the search space, which means that, unlike the use of local search methods, we can still determine inconsistencies. The bookkeeping mechanism involves keeping track of some additional information for each variable in the stack, namely which assignments have been tried thus far. This can be done relatively inexpensively.

Randomization of the look-ahead and look-back procedures can also be done in a relatively straightforward manner, for example, by randomizing the decision of whether to apply such procedures after each variable/value assignment. This technique can be useful, especially when look-ahead techniques are expensive. On the other hand, randomization of the backtrack points requires the use of data structures that are more complicated in general, so there may be a substantial increase in the time/space requirements of the algorithm. Lynce *et al.*, (Lynce et al., 2001; Lynce and Marques-Silva, 2002b) discuss learning strategies and randomization of the backtrack points (specifically, random backtracking and unrestricted backtracking), as well as how to maintain completeness (see also section 6.6).

We should note that when implementing a randomized algorithm, one uses a pseudo random-number generator, which is in fact a deterministic algorithm for generating "random" numbers. By choosing different initial random seeds, we obtain different runs of the algorithm. For experimental purposes, it is important to save the "seed" given to the random-number generator, so that the same experiment can be replayed.

One can also speak of "deterministic randomization" (Wolfram, 2002), which expresses the fact that the behavior of some very complex deterministic systems is so unpredictable that it actually appears to be random. This notion of deterministic randomization is implicitly used, for example, in calculating "random" backtrack points by applying a deterministic formula to the clauses learned during search (Lynce et al., 2001; Lynce and Marques-Silva, 2002b). Another example is the restarting of (deterministic) backtrack search solvers that feature clause learning: Each time the solver is restarted, it behaves quite differently from the previous run (because of the additional learned clauses), and thus appears to behave "randomly" (Moskewicz et al., 2001; Lynce and Marques-Silva, 2002b).

As mentioned earlier, the performance of a randomized backtrack search algorithm can vary dramatically from run to run, even on the same instance. As a consequence, runtime distributions of backtrack search algorithms are often characterized by heavy tails. In the next section we provide formal models characterized by such non-standard distributions.

## 3. Formal Models of Heavy-Tailed Behavior

In order to provide an intuitive feel for the notion of heavy-tailedness, in this section we present several applications for which formal results demonstrate heavy-tailed behavior of the underlying distributions. We start by introducing the random-walk model, also known as the aimless wanderings of a "drunkard," which is characterized by an infinite mean. We then describe a tree-search model and indicate which values of the relevant parameters yield both an infinite mean and an infinite variance, and which yield just an infinite variance. We also discuss the notion of *bounded* heavy tails within the context of the tree-search model.

### 3.1 Random Walk

Consider a one-dimensional, symmetric random walk, where at each time step one takes a unit step up or down with equal probability (Feller, 1968; Feller, 1971; see figure 1.2 (top panel), where the quantities plotted on the horizontal and vertical axes are the number of time steps and the distance from the origin, respectively, and both are in units of 1000). One can show that after starting at the origin, with probability 1, the walk will eventually return to the origin. The *expected* time before return is infinite, however, and on average a walk of this sort will reach all possible points before its return. Another intriguing phenomenon involves the expected number of returns to the origin ("zero-crossings") in a given number of steps. Intuition would dictate that if in $k$ steps one has $l$ crossings on average, then in a walk that is $m$ times as long, one would expect on average $m \times l$ crossings. It can be shown, however, that in $m \times k$ steps, one will observe only $\sqrt{m} \times l$ crossings, on average. This means that there can be surprisingly long periods between crossings (see figure 1.2 (top panel); note, for example, the long period between step 2544 and step 7632, during which time there is no crossing). In fact, when doing a series of $r$ random walks, each terminating at the first crossing, some of the walks will be of the same order as the length of all the other walks combined (on average), *no matter what the value of* $r$ *is.* Such events would normally be dismissed as outliers, but when dealing with heavy-tailed distributions, they are far from rare and are an inherent aspect of the distribution. These distributions are, therefore, good models for dealing with phenomena that exhibit extreme fluctuations.

A visual illustration of the heavy-tail behavior of the runtime distribution of a random walk is depicted in the bottom panel in figure 1.2. To generate this distribution we used simulation data for 10,000 runs of a symmetric random walk. For each run we recorded the number of steps $X$ it took to return to the origin for the first time. If $x$ is the number of time steps , let $f(x)$ denote

the empirical probability that the walk returned to the origin in *exactly* $x$ steps, and let $F(x)$ be the (cumulative) probability of returning in *at most* $x$ steps (i.e., $f(x) = \Pr\{X = x\}$ and $F(x) = \Pr\{X \leq x\}$). In the figure, we plot the complement-to-one of the cumulative distribution, namely, $1 - F(x)$, which is just the probability that the walk did *not* return to the origin within $x$ steps. (Note that $1 - F(x) = 1 - \Pr\{X \leq x\} = \Pr\{X > x\}$.) Of course, none of the runs could have returned to the origin after just one step, so $f(1) = 0$. Moreover, $f(2) = 0.5$, so we have $F(2) = 0.5$, which means that in our experiment the walk had a 50% chance of returning to the origin in at most two steps.

In the figure, we give a log–log plot of $1 - F(x)$. (In the figure, the walk data is given by the diagonal straight line.) As the figure shows, we obtain a nearly straight line for the tail of the distribution. This suggests that the function $1 - F(x)$ has power-law decay, *i.e.*, we have $1 - F(x) = \Pr\{X > x\} \sim Cx^{-\alpha}$, and thus the distribution is heavy tailed (see section 4.1.2). The slope of the line gives us an estimate of the index of stability, $\alpha$, which in this case is equal to 0.5 (also known by rigorous analysis). The relatively high frequency of large outliers is clear from the figure. For example, although 50% of the walks return in just 2 steps, 1% of the walks take more than 5,000 steps to return to the origin, and about 0.1% take over 200,000 steps. In fact, several of the walks in our sample take almost $1,000,000$ steps.

To demonstrate how different such a heavy-tailed distribution is from a standard distribution, we include in the figure a plot of the complement-to-one of the cumulative distribution of a normal distribution, using simulation data and a mean value of 2. We present curves for two different values of the standard deviation: $\sigma = 1$ (left-most curve) and $\sigma = 10^6$ (right-most curve). The key property to observe is the sharp, faster-than-linear decay of the normal distribution in the log–log plot, which is consistent with the exponential decay in the tail of the distribution. We included a normal distribution with $\sigma = 10^6$ to show that the drop-off in the tail remains sharp even when the normal distribution has a large standard deviation. (The normal distribution is symmetrical; the figure gives only the right-hand side.)

## 3.2 Tree Search Model

Intuitively, heavy-tailed behavior in backtrack-style search arises from the fact that wrong branching decisions may lead the procedure to explore an exponentially large subtree of the search space that contains no solutions. Depending on the number of such "bad" branching choices, one can expect considerable variation in the time to find a solution from one run to another. Heavy-tailed behavior has been shown not to be inherent to backtrack search in general,

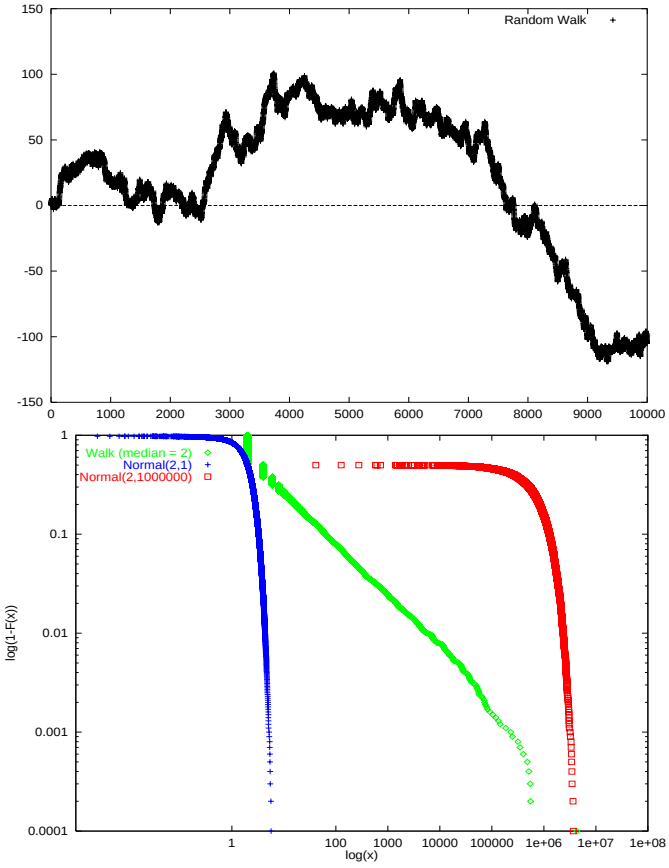



*Figure 1.2.* Top panel: symmetric random walk (10,000 steps). Bottom panel: the heavy-tailed nature of a random walk contrasted with the exponential decay of the normal distribution.

but rather to depend on the structure of the underlying tree search and on the *pruning* power of the heuristics (Chen et al., 2001).

In our analysis, we contrast two different tree-search models: a balanced model that does not exhibit heavy-tailed behavior, and an *imbalanced* model. A key component of the imbalanced model is that it allows for highly irregular and imbalanced trees, which are radically different from run to run. For the imbalanced tree-search model, we formally show that the runtime of a randomized backtrack search method is heavy tailed for a range of values of the model parameter $p$, which characterizes the effectiveness of the branching heuristics and pruning techniques. The heavy-tailedness leads to a runtime distribution with an infinite variance, and sometimes an infinite mean. The imbalanced tree model is depicted in the top panel in figure 1.3.

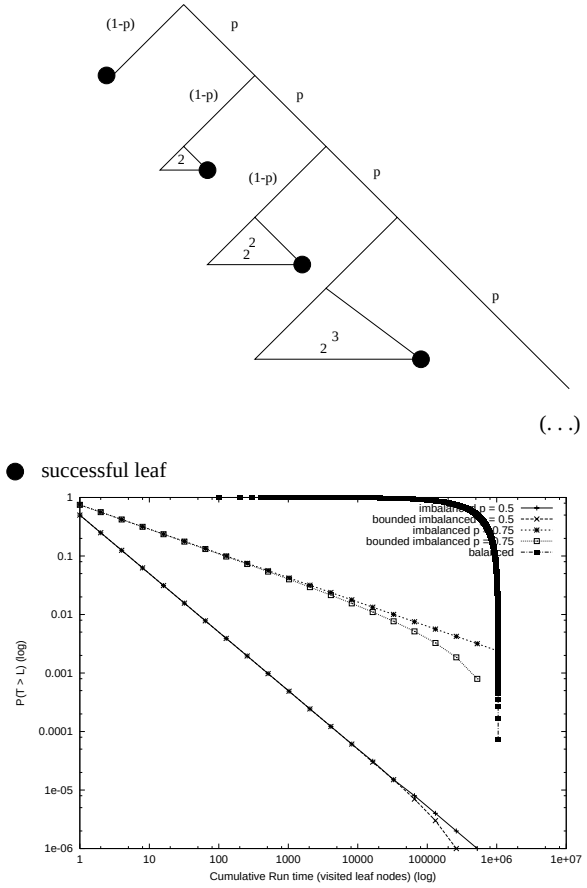



*Figure 1.3.* Top panel: imbalanced tree model; $p$ is the probability that the branching heuristic will make a wrong decision, and $b$ is the branching factor (in this case, $b = 2$). This tree model has a finite mean and an infinite variance when $1/b^2 < p < 1/b$; both the mean and the variance are infinite when $p \geq 1/b$. Bottom panel: example distributions the imbalanced and bounded imbalanced models, contrasted with the balanced model. Parameters: $b = 2$, $n = 20$, $p = 0.5$ and $0.75$ ($n$ is the number of variables).

The imbalanced tree model assumes that the probability that the branching heuristic will make a wrong decision is $p$, hence that, with probability $(1-p)$ the search is guided directly to a solution. With probability $p(1-p)$, a search space of size $b$, with $b \geq 2$, needs to be explored. In general, with probability $p^j(1-p)$, a search space of $b^j$ nodes needs to be explored. Intuitively, $p$ is the probability that the overall amount of backtracking increases geometrically by a factor of $b$. This increase in backtracking is modeled as a global phenomenon. The larger $b$ and $p$ are, the "heavier" the tail. Indeed, when $b$ and $p$ are sufficiently large, so

that $bp \geq 1$, the expected value of $T$ (where $T$ is the runtime of the algorithm)[1] is infinite: $E[T] = \infty$. If, however, $bp < 1$ ("better search control"), we obtain a finite mean of of $E[T] = (1-p)/(1-pb)$. Similarly, if $b^2p > 1$ the variance of $T$ is infinite; otherwise, it is finite.

**Bounded Heavy-Tailed Behavior for Finite Distributions.**

The imbalanced tree-search model does not put an *a priori* bound on the size of the search space. In practice, however, the runtime of a backtrack search procedure is bounded above by some exponential function of the size of the instance. We can adjust our model by considering heavy-tailed distributions with bounded support — "bounded heavy-tailed distributions", for short (see *e.g.*, Harchol-Balter et al., 1998). Our analysis of the bounded case shows that the main properties of the runtime distribution observed for the unbounded, imbalanced search model have natural analogues when dealing with finite but exponential-sized search spaces. Heavy-tailed distributions have infinitely long tails with power-law decay, while bounded heavy-tailed distributions have exponentially long tails with power-law decay. Also, the concept of an infinite mean in the context of a heavy-tailed distribution translates into a mean that is exponential in the size of the input, when considering bounded heavy-tailed distributions.

The bottom panel in figure 1.3 illustrates and contrasts the distributions for the various models. We use a log-log plot of the tails of the various distributions to highlight the differences between them. The linear behavior over several orders of magnitude for the imbalanced models is characteristic of heavy-tailed behavior (see section 4.1.2). The drop-off at the end of the tail of the distribution for the bounded case illustrates the effect of the boundedness of the search space. However, given the relatively small deviation from the unbounded model (except at the very end of the distribution), we see that the effect of bounding is relatively minor. In the plot we also contrast the behavior of the imbalanced model that of a balanced model (for details, see Chen et al., 2001). The sharp drop-off for the balanced model indicates the absence of heavy-tailedness.

The heavy-tailedness of both of the imbalanced tree-search models occurs as a consequence of the competition between two critical factors: an exponentially increasing penalty in the size of the space to be searched as the number of "mistakes" caused by the branching heuristic increases, and an exponentially decreasing probability that a series of branching mistakes will be made.

[1] $T$ is the number of leaf nodes visited, up to and including the successful leaf node.

## 4. Heavy and Fat-Tailed Distributions

We first identified the phenomenon of heavy tails in combinatorial search in a study of the quasigroup (or Latin-square) completion problem (QCP) ( Gomes et al., 1997; Gomes et al., 2000). A Latin square, which is an abstract structure from finite algebra, is an NxN table on N symbols in which each symbol occurs exactly once in each row and column. A Latin square with N rows corresponds to the multiplication table of a quasigroup of order N. QCP consists in determining whether a partially filled Latin square can be completed in such a way that we obtain a full Latin square (see figure1.4). QCP is NP-complete ( Colbourn, 1984). QCP was introduced as a benchmark problem for evaluating combinatorial search methods, since its structure is similar to that found in real-world problems such as scheduling, timetabling, routing, and design of statistical experiments. In fact, QCP directly maps into the problem of fiber-optics routing, which explains the use of the term *Latin Routers* (Laywine and Mullen, 1998; Kumar et al., 1999).

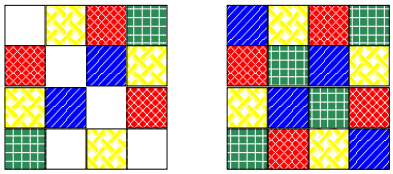


*Figure 1.4.* Quasigroup Completion Problem (QCP).

In our study, we encoded QCP as a constraint satisfaction problem (CSP), associating a different variable with each cell of the Latin square (Gomes and Selman, 1997b). Given a partial Latin square of order $N$, QCP can be expressed as follows:

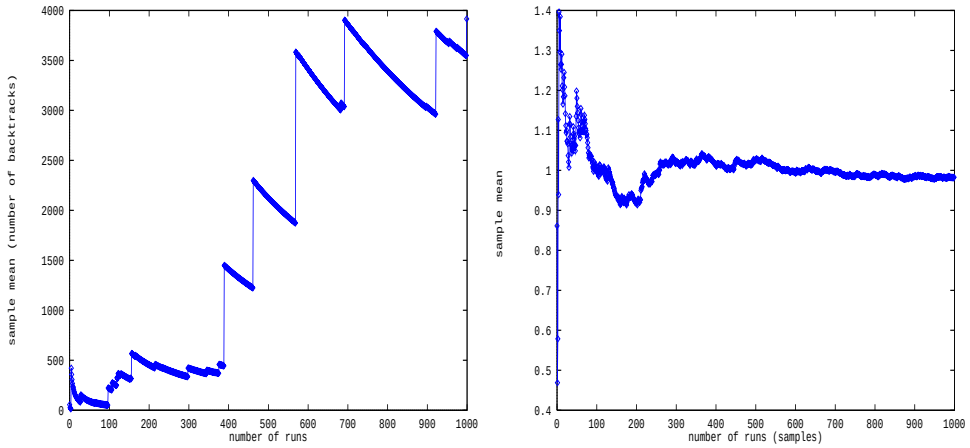
$$x_{i,j} \in \{1, \ldots, N\} \quad \forall i, j$$
$$x_{i,j} = k \quad \forall i, j \text{ such that QCP}_{ij} = k$$
$$\texttt{alldiff}\,(x_{i,1}, x_{i,2}, \ldots, x_{i,n}), \quad i = 1, 2, \ldots, n$$
$$\texttt{alldiff}\,(x_{1,j}, x_{2,j}, \ldots, x_{n,j}), \quad j = 1, 2, \ldots, n.$$

$x_{i,j}$ denotes the symbol in cell $i, j$, and the statement "QCP$_{ij} = k$" denotes that symbol $k$ is pre-assigned to cell $i, j$.

The `alldiff` constraint states that all the variables involved in the constraint have to have different values.

For our experiments we used a randomized backtrack search procedure, with the so-called first-fail heuristic combined with forward checking. In the first-fail heuristic, the next variable to branch on is the one with the smallest remaining

domain, *i.e.*, the search procedure chooses to branch on the variable with the fewest possible options left to explore, thereby leading to the smallest branching factor. In case of a tie, the standard approach is to break the tie using lexicographical order. Since we were interested in analyzing the variance of the backtrack search procedure on a particular instance, *isolating it from the variation that results from considering different instances*, we simply replaced the standard (lexicographical) order for tie-breaking with random tie-breaking. With this change, we obtained a *randomized* backtrack search procedure. In other words, each run of our randomized backtrack search algorithm on the *same* instance may vary in the order in which choices are made —- and, potentially, in solution time as well. Again, we note that introducing randomness into the branching-variable/value selection does not affect the completeness of the backtrack search.




(a) (b)

*Figure 1.5.* Erratic behavior of mean cost of completing a quasigroup (order 11, 30% preassignment) vs. stabilized behavior of mean for a standard distribution (gamma)

In order to obtain *uncensored* empirical runtime distributions for our backtrack search procedure, *i.e.*, without having to interrupt any run, we considered relatively easy QCP instances, of order 11. The phase transition in solvability of QCP, for instances of order 11, occurs when approximately 42% of the cells of the Latin square are prefilled, with a corresponding peak in complexity ( Gomes and Selman, 1997b). In our experiments, we sampled instances from the under-constrained area (30%). For each empirical runtime distribution, we typically performed at least $1,000$ runs of the randomized backtrack search procedure on the same instance.

Figure 1.5(a) shows the mean cost of our randomized backtrack-style search procedure on our QCP instance of order 11 with 30% pre-assignment, calculated over an increasing number of runs. The figure illustrates some of the puzzling features of randomized complete search procedures, such as extreme variability in (and a seemingly wandering value of) the sample mean. In other words, the mean of the distribution exhibits highly erratic behavior that does not stabilize with increasing sample size. Again, we note that this instance was relatively *easy*. In fact, $60\%$ of the runs took no more than *one* backtrack. (The median, not shown in the figure, is 1, and it stabilizes rather quickly.) Contrast this behavior with that of the mean of the standard probability distribution[2] given in figure 1.5(b). In this plot, which depicts a gamma distribution, we see that the sample mean converges rapidly to a constant value as the sample size increases.

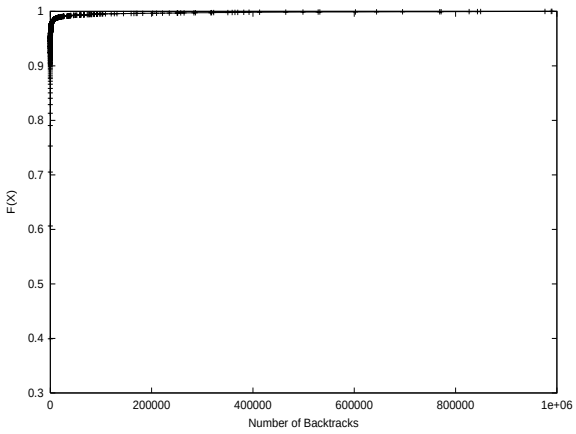



*Figure 1.6.* The phenomenon of long tails with randomized complete search procedures (QCP instance, order 11, 30% pre-assignment)

Figure 1.6 provides some insight into the wandering-mean phenomenon. The figure shows the surprisingly long "tail" of the distribution. The curve gives the empirical cumulative distribution of the search cost (measured in number of backtracks). It was produced by running the randomized backtrack search procedure 1,000 times on the same instance, namely, the one shown in figure 1.5 (a) (order 11, 30% pre-assignment). Even though 60% of the runs were solved with no more than one backtrack, $0.5\%$ of them were still not solved after $100,000$ backtracks, and some runs took over $1,000,000$ backtracks.

In order to model the long tails of runtime distributions of randomized backtrack search we consider *heavy-tailed* distributions.

[2]By a standard distribution we mean a distribution that has all its moments defined and finite.

## 4.1 Heavy-Tailed Distributions

Heavy-tailed distributions were first introduced by the Italian-born Swiss economist Vilfredo Pareto in the context of income distribution. They were extensively studied mathematically by Paul Lévy in the period between the world wars. Lévy worked on a class of probability distributions with heavy tails, which he called *stable* distributions. At the time, however, these distributions were largely considered probabilistic curiosities or pathological cases, mainly used as counterexamples. This situation changed dramatically with Mandelbrot's work on fractals. In particular, two seminal papers by Mandelbrot were instrumental in establishing the use of stable distributions for modeling real-world phenomena (Mandelbrot, 1960; Mandelbrot, 1963).

Recently, heavy-tailed distributions have been used to model phenomena in areas as diverse as economics, statistical physics, and geophysics. In particular, they have been applied to stock-market analysis, Brownian motion, weather forecasting, and earthquake prediction — and even for modeling of time delays on the World Wide Web (see *e.g.*, Adler et al., 1998; Mandelbrot, 1983; Samorodnitsky and Taqqu, 1994).

### 4.1.1 Definition.

Heavy-tailed distributions have tails that are asymptotically of the Pareto-Lévy form:

$$\Pr\{X > x\} \sim Cx^{-\alpha}, \quad x > 0. \tag{1.1}$$

where $\alpha$ is a positive constant. These are distributions whose tails exhibit a *hyperbolic* (i.e., slower than exponential) decay. In the case treated here, it suffices to consider this tail behavior for only positive values of the random variable $X$, so in what follows we will assume that the distribution is supported on only the positive line (hence that $\Pr\{0 \leq X < \infty\} = 1$).

The constant $\alpha$ is called the *index of stability* of the distribution, because *which* moments of $X$ (if any) are finite is completely determined by the tail behavior. Note that $\alpha = \inf\{r > 0 : E[X^r] = \infty\}$; hence all the moments of $X$ which are of order strictly *less than* $\alpha$ are finite, while those of higher order ($\geq \alpha$) are infinite. For example, when $\alpha = 1.5$, the distribution has a finite mean but not a finite variance. With $\alpha = 0.6$, the distribution has neither a finite mean nor a finite variance.

Mandelbrot, 1983, provides an excellent introduction to heavy-tailed distributions, with a discussion of their inherently self-similar or fractal nature. Many aspects of random walks involve heavy-tailed distributions (see *e.g.*, Feller, 1968; Feller, 1971). Adler et al., 1998, provide a collection of essays concerning techniques and approaches for the analysis of heavy-tailed distributions, as well as applications of heavy-tailed modeling (*e.g.*, in the areas of telecommu-

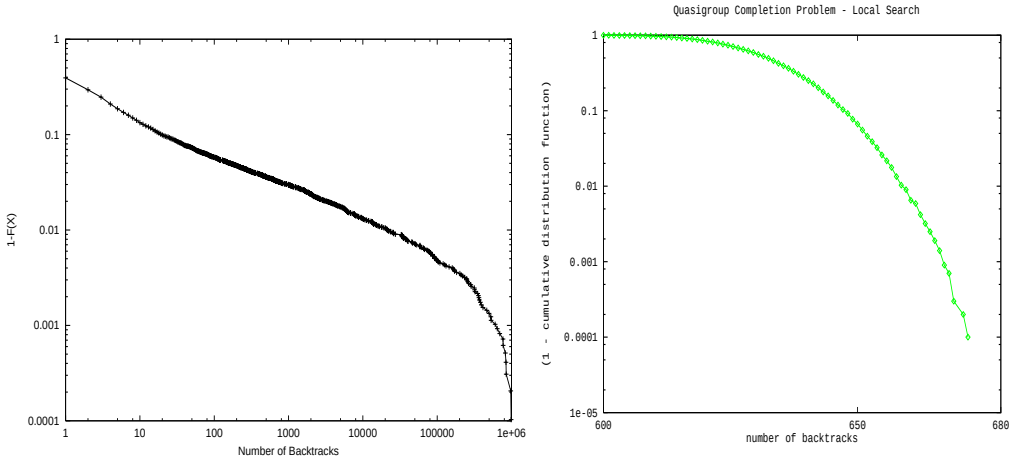
nications, Web, and finance). Heavy-tailed distributions are closely related to stable distributions. For a complete treatment of stable distributions, see either Zolotarev, 1986, or the more modern approach of Samorodnitsky and Taqqu, 1994.

### 4.1.2 Visualization of Heavy-Tailed Behavior.

In order to check for the existence of heavy tails in our distributions, we start by plotting and analyzing the behavior of the tails.

If a Pareto-Lévy tail is observed, then the rate of decrease of the estimated probability density of a distribution is hyperbolic — *i.e.*, slower than exponential. The complement-to-one of the cumulative distribution $1 - F(x)$ $(= \Pr\{X > x\})$, also displays hyperbolic decay. By equation 1.1, we have $1 - F(x) \sim Cx^{-\alpha}$.

Given the hyperbolic decay of the complement-to-one of the cumulative distribution of a heavy-tailed random variable, the tail of a log–log plot of its empirical distribution should show an *approximately linear decrease*; moreover, the slope of the observed linear decrease provides an estimate of the index $\alpha$. This is in contrast to a distribution that decays exponentially, where the tail of a log–log plot should show a faster-than-linear decrease. Figure 1.7(a) displays




(a) (b)

*Figure 1.7.* Left panel: log–log plot of heavy-tailed behavior (QCP instance, order 11, 30% pre-assignment; uncensored data). Right panel: no heavy-tailed behavior.

a log–log plot of the complement-to-one of the cumulative distribution for our QCP data (order 11, 30% pre-assignment). The linear nature of the tail of this log–log plot extends over several orders of magnitude, which strongly indicates that the tail is of the Pareto-Lévy type. We were able to obtain the data for this

plot by running our backtrack search procedure without a *cutoff* in the number of backtracks — *i.e.*, the data were *uncensored*. (The small drop-off at the end of the tail corresponds to less than 0.2% of the data, and thus is not statistically significant; see also section 3.2.)

By way of comparison, in figure 1.7(b) we show a log–log plot of the complement-to-one of the cumulative distribution of a randomized *local* search procedure on a harder quasigroup instance. (Local search procedures solve our original instance — of order 11, with 30% pre-assignment — trivially.) It is clear from the plot that the distribution for the harder instance does not exhibit heavy-tailed behavior, given the faster-than-linear decay of the tail. (See also section 5.)

### 4.1.3 Estimation of Index of Stability ($\alpha$).

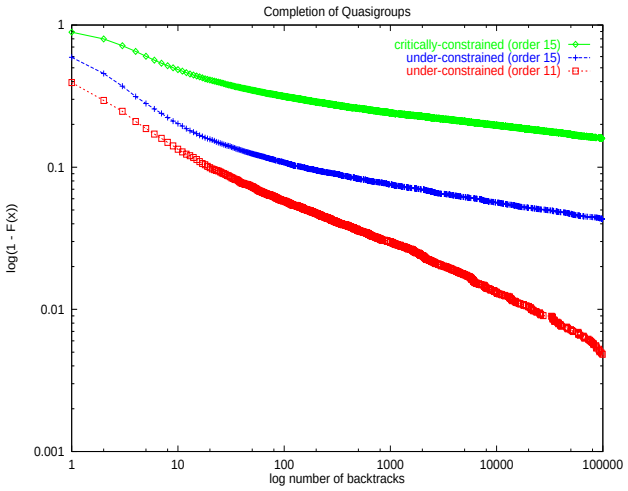



*Figure 1.8.* Log-log plot of heavy-tailed behavior.

To supplement the visual check of heavy-tailed behavior, we estimate the index of stability, $\alpha$.

There is a standard estimator, called the Hill estimator (Hill, 1975; Hall, 1982), for determining the index of stability. Quite often, however, we cannot use this analytical tool, since the final outcomes of some of our runs are not observable: In practice, quite often we have to run the experiments with a certain *high cutoff* in the number of backtracks (*i.e.*, we have to resort to the use of *censored* data), to avoid literally getting stuck in the most extreme outliers in the tail of the distribution during our data collection. The selection of the cutoff depends on the problem instance and the algorithm. Obviously, the higher the better. In general, we try to gather data over a fairly sizable range in the value

| Cases | $k$ | $u$ | $\alpha$ |
|---|---|---|---|
| Quasigroup (order 11, 30%) | 10,000 | 0 | 0.466 (0.009) |
| Quasigroup (order 15, 30%) | 10,000 | 431 | 0.319 (0.006) |
| Quasigroup (order 15, 40%) | 10,000 | 1933 | 0.153 (0.003) |

*Table 1.1.* Estimates of the index of stability ($\alpha$). $k$ is the sample size, and $u$ is the number of runs that failed to terminate before reaching the cutoff in the number of backtracks. The values within parentheses are the estimated asymptotic standard deviations.

of the random variable, ideally over at least 5 to 6 orders of magnitude (i.e., with the number of backtracks in the range $10^5$–$10^6$), in which case we see some indication of stabilization of the tail behavior. Ideally, we would like not to exclude a significant portion of the tail, but sometimes this is just too costly from a computational point of view.

Figure 1.8 depicts the tails of one critically constrained QCP instance (order 15, 40% pre-assignment) and two solvable under-constrained QCP instances (one being of order 15, with 30% pre-assignment, and the other being the same instance of order 11, with 30% pre-assignment, that was used in figures 1.5(a), 1.6, and 1.7). The computational cost for the instances of order 15 was so high that we were forced to censor the runs, with a cutoff of $10^5$ backtracks.

We adapted the Hill estimator for application to a truncated set of data by deriving the following maximum-likelihood estimator for the index of stability $\alpha$ (for details, see Gomes et al., 2000):

$$\hat{\alpha}_{r,u} = \left( \frac{1}{r} \sum_{j=1}^{r-1} \ln X_{n,n-r+j} + \frac{u+1}{r} \ln X_{n,n} - \frac{u+r}{r} \ln X_{n,n-r} \right)^{-1} \tag{1.2}$$

where $n$ is the number of runs that terminate within the maximum number of backtracks allowed in the experiment, $n+u$ is the total number of runs, and $X_{n,1} \leq X_{n,2} \leq \cdots \leq X_{n,n} \leq X_{n,n+1} \leq \cdots \leq X_{n,n+u}$ is a numerical ordering of the observations in the sample (*i.e.,* the observed numbers of backtracks for different runs of our procedure). Because of the imposition of the cutoff, the highly extreme values $X_{n,n+1}, X_{n,n+2}, \ldots, X_{n,n+u}$ are not observable. The parameter $r$ (which is less than $n$) translates into a lower-bound cutoff on search cost, which enables us to focus only on the tail end of the distribution (note that only the observations $X_{n,n-r} \leq X_{n,n-r+1} \leq \cdots \leq X_{n,n}$ enter into the estimation of $\alpha$). If all the runs terminate (the $u = 0$ case), the above estimator reduces to the Hill estimator.

Table 1.1 displays the maximum-likelihood estimates of the index of stability (the values of $\alpha$) for the QCP instances in figure 1.8. Note that for each of the instances shown in the table, the estimated value of $\alpha$ is consistent with the hypothesis of an infinite mean and an infinite variance, since $\alpha \leq 1$.

| c | Normal | Cauchy | Lévy |
|---|---|---|---|
| 0 | 0.5000 | 0.5000 | 1.0000 |
| 1 | 0.1587 | 0.2500 | 0.6827 |
| 2 | 0.0228 | 0.1476 | 0.5205 |
| 3 | 0.001347 | 0.1024 | 0.4363 |
| 4 | 0.00003167 | 0.0780 | 0.3829 |
| 5 | 0.0000002866 | 0.0628 | 0.3453 |

*Table 1.2.* Comparison of tail probabilities, $\Pr\{X > c\}$, for standard normal, Cauchy, and Lévy distributions

## 4.2 Fat *vs.* Heavy-Tailed Distributions

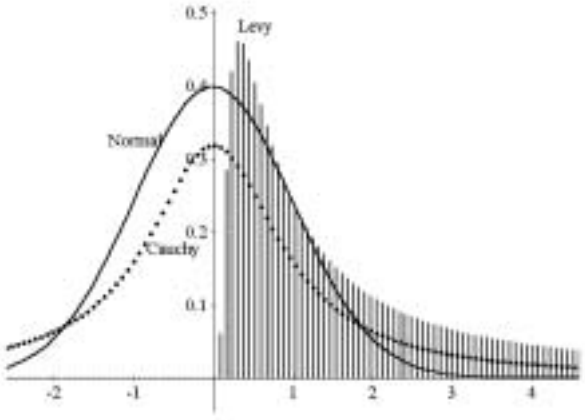



*Figure 1.9.* Standardized normal, Cauchy, and Lévy densities

The attribution of the name "heavy tails" to this class of distributions definitely captures their main feature, namely, the large proportion of the total "probability mass" which is concentrated in the tail. This high concentration of probability mass is reflected in the slow decay of the tail and is responsible for the fact that *all the moments of a heavy-tailed distribution from some order on are infinite.* The heavier the tail, the lower the order at which the onset of the infinite moments occurs. Distributions with very heavy tails have an infinite mean (first moment), while those with "lighter" tails have a finite mean—and some have finite moments for one or more of the higher orders as well

Figure 1.9 contrasts three distributions: standard normal, Cauchy, and Lévy. The key property to observe is the dramatic difference in the decay of the

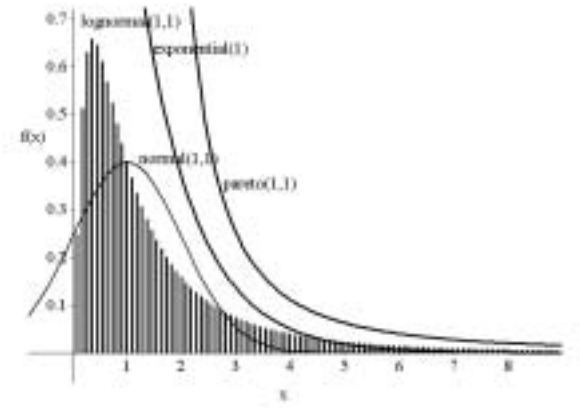



*Figure 1.10.* Comparison of the *fatness* of tails: normal (1,1), lognormal(1,1), exponential(1), and pareto(1,1).

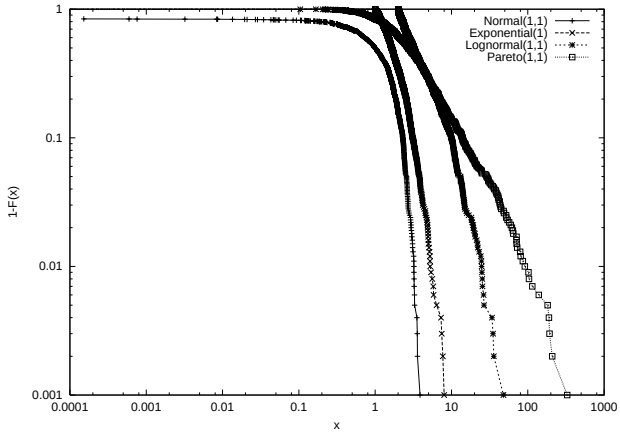



*Figure 1.11.* Comparison of the tail *fatness*: normal (1,1), lognormal(1,1), exponential(1), and pareto(1,1).

tails of the distributions. In table 1.2, we present the proportion of the total probability mass in the tail of each of these distributions for various values of the random variable. It is clear that this quantity quickly becomes negligible for the standard normal distribution, whereas the other two distributions have a significant proportion of the total probability mass in their tails. The lower the value of $\alpha$ ( the *index of stability* of the distribution), the heavier the tail. For example, the Cauchy distribution has $\alpha = 1.0$, and the Lévy distribution has $\alpha = 0.5$.

Related to *heavy*-tailedness is the notion of *fat*-tailedness, which is defined in terms of a property of a distribution known as the *kurtosis*.

The *kurtosis* of a distribution is the quantity $\mu_4/\mu_2^2$, where $\mu_2$ and $\mu_4$ are the second and fourth moments about the mean, respectively (note that $\mu_2$ is the variance). The kurtosis is independent of the location and scale parameters of a distribution. If a distribution has a high central peak and long tails, then in general the kurtosis is large.

The kurtosis of the standard normal distribution is 3. A distribution with a kurtosis larger than 3 is said to be *fat-tailed* or *leptokurtic*. Like a *heavy*-tailed distribution, a *fat*-tailed distribution has a long tail and a considerable concentration of mass in its tail. Nevertheless, the tail of a *fat*-tailed distribution decays at a faster rate than that of a heavy-tailed distribution. In fact, all the moments of a fat-tailed distribution are finite, which is in stark contrast to the moments of a heavy-tailed distribution.

Examples of distributions that are characterized by fat tails are the exponential, lognormal, and Weibull distributions. In figure 1.10 we compare the tails of four distributions: normal (1,1), lognormal (1,1), exponential (1), and Pareto (1,1). Clearly, the tail of the Pareto distribution, a heavy-tailed distribution with $\alpha = 1$ (and therefore an infinite mean), is the heaviest. The tail of the lognormal distribution is fatter than that of the exponential, and both of them have fat tails —fatter, in fact, than the tail of the normal distribution.

Figure 1.11 shows a log–log plot of the tails of the same four distributions. (For each one, we generated 1000 data points from the corresponding (theoretical) distribution.) The relative fatness of the tails becomes more apparent in this plot. Note the qualitatively different (and approximately linear) behavior of the drop-off in the Pareto tail, which is suggestive of the heavy-tailed nature of the Pareto distribution.

As a final note, we should mention that there are a number of different parametric models for performing extreme-value analysis, each of which is suited to a certain class of distributions. We have already mentioned fat-tailed and heavy-tailed distributions, such as lognormal and sum-stable distributions, respectively. Extreme-value (EV) and generalized Pareto (GP) models are also central to the statistical analysis and modeling of extremes. A treatment of such models is beyond the scope of this book. (For further information on this topic, see, *e.g.*, Adler et al., 1998; Mandelbrot, 1983; Reiss and Thomas, 2001; Samorodnitsky and Taqqu, 1994.)

## 5. Heavy and Fat-Tailed Distributions in Backtrack Search

The phenomena of *heavy* and *fat* tails in randomized backtrack search have been observed in several domains other than QCP, and in considering different

search paradigms, namely, general constraint satisfaction problem (CSP) formulations and mixed integer programming (MIP) formulations. They have also been observed when considering more specific problem formulations such as Boolean satisfiability (SAT), graph coloring (GC), and theorem proving (TP). Theorem proving is beyond the scope of this book. (See *e.g.*, Meier et al., 2001, for a description of results on heavy-tailed behavior in proof planning, a particular area of theorem proving.)

## 5.1 CSP Formulations

For the QCP experiments, we incorporated randomization into a CSP approach. Specifically, we used the Ilog constraint-solver engine, which provides a powerful C++ constraint programming library (Ilog, 2001b). As described in section 4, we randomized the first-fail heuristic. Not only did we consider several variants of this heuristic, by randomizing variable and/or value selection, but we considered a popular extension of it — namely, the Brelaz heuristic ( Brelaz, 1979), which was originally introduced for graph coloring procedures.

The Brelaz heuristic specifies a way for breaking ties that are encountered on applying the first-fail rule: If two variables have equally small remaining domains, the Brelaz heuristic chooses the variable that shares constraints with the largest number of the remaining unassigned variables. Any ties that remain after the Brelaz rule are resolved randomly. As with the first-fail heuristic, we studied several variants of the Brelaz heuristic, by randomizing variable and/or value selection. Though the variants of the Brelaz heuristic proved more powerful than those of the first-fail heuristic, the runtime distributions were qualitatively similar. One significant difference between the two was that in using the Brelaz heuristic, we observed heavy tails, similar to those in figure 1.8.

Sports scheduling is another domain in which heavy-tailed behavior has been observed, using a CP paradigm (Gomes et al., 1998b). There has been an increasing interest in applying CP techniques to sports-scheduling problems, given their highly combinatorial nature (Easton et al., 2001; McAloon and Tretkoff, 1997; Nemhauser and Trick, 1998; Regin, 1999). In sports scheduling problems, one of the issues is timetabling, where one has to take into consideration constraints on how the competing teams can be paired, as well as how each team's games are distributed over the entire schedule. In particular, we consider the timetabling problem for a "round-robin" schedule: Every team must play every other team exactly once. The *global* nature of the pairing constraints makes this a particularly hard combinatorial search problem. Typically, a game will be scheduled on a certain field or court, at a certain time, etc. This kind of combination will be called a slot. These slots can vary in desirability with respect to such factors as lateness in the day, location, and condition of the field.

The problem is to schedule the games in such a way that the different periods are assigned to the teams in an equitable manner over the course of the season.

The round-robin sports-scheduling problem considered here is something of a "classic" in the operations research community, because it presents a very tough challenge for integer programming methods. An instance of this problem of size 10, encoded as an integer programming problem, is included in the OR-Library (Beasley, 1990).

In order to study the runtime distributions of randomized backtrack search on this benchmark problem, we considered a problem instance of size 12, even though at the time of these experiments the best algorithm could solve instances of size up to 14 teams.[3] Such an instance would have required a tremendous amount of computational resources, making it difficult for us to obtain a good empirical distribution without eliminating a large portion of the tail. In fact, even for 12 teams, we had to use a cutoff of 250,000 backtracks per run. We randomized the backtrack solver, randomly breaking ties on variable/value selection.

Figure 1.12 shows a log–log plot of the complement-to-one of the cumulative distribution, 1-F(x), for our round-robin sports-scheduling problem ($N = 12$). Plot (a) gives the full range of the distribution, while plot (b) shows only the tail ($X > 10,000$). The linear nature of the tail, which is magnified in plot (b), strongly suggests that it is of the Pareto-Lévy type. (Our cutoff truncated only about 0.2% of the tail.)

## 5.2 Mixed Integer Programming Formulations

The standard approach used by the OR community to solve mixed integer programming problems (MIP) is branch-and-bound search. First, a linear program (LP) relaxation of the problem instance is considered. In such a relaxation, all variables of the problem are treated as continuous variables. If the solution to the LP relaxation problem has non-integer values for some of the integer variables, we have to branch on one of those variables. This way we create two new subproblems (nodes of the search tree), one with the floor of the fractional value and one with the ceiling. (For the case of binary (0/1) variables, we create one instance with the variable set to 0 and another with the variable set to 1.) The standard heuristic for deciding which variable to branch on is based on the degree of infeasibility of the variables ("max-infeasibility variable selection"): We select the variable whose non-integer part in the solution of the LP relaxation is closest to 0.5. Informally, we pick the variable whose value is "furthest away from an integer."

[3] CP techniques can now solve the round-robin sports-scheduling problem up to 40 teams (Regin, 2002).

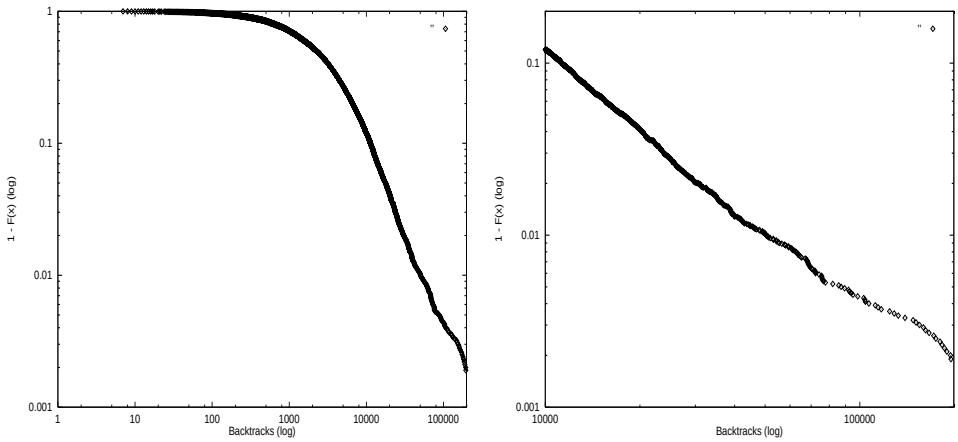



*Figure 1.12.* Log-log plot of the complement to one of the cumulative distribution of the runtime for the round-robin sports scheduling (12 teams; CSP formulation); (a) full distribution; (b) tail ($X > 10,000$).

Following the strategy of repeatedly fixing integer variables to integer values will lead at some point to a subproblem with an overall integer solution, provided we are dealing with a feasible problem instance. (We call any solution where all the integer variables have integer values an "integer solution.") In practice, it often happens that the solution of the LP relaxation of a subproblem already *is* an integer solution, in which case we do not have to branch further from this node.

Once we have found an integer solution, its objective function value can be used to prune other nodes in the tree, namely, those whose relaxations have worse values. This is because the LP relaxation bounds the optimal solution of the problem. For example, for a minimization problem, the LP relaxation of a node provides a lower bound on the best possible integer solution.

A critical issue that determines the performance of branch-and-bound is the way in which the next node to be expanded is selected. In OR, The standard approach is to use a best-bound selection strategy. That is, from the list of nodes (subproblems) to be considered, we select the one with the best LP bound. (This approach is analogous to an A$^\star$-style search. The LP relaxation provides an admissible search heuristic.)

The best-bound node-selection strategy is particularly well-suited for reaching an optimal solution (because of the greedy guidance), which has been the traditional focus of much of the research in OR. However, one significant drawback of this approach is that it may take a long time before the procedure finds an integer solution, because of the breadth-first flavor of the search. Also, the

approach has serious memory requirements because the full fringe of the tree has to be stored.

Given problems that have a difficult feasibility part, the best-bound approach may take too long before reaching an integer solution. (Note that an integer solution is required before any nodes can be pruned.) In our experiments, therefore, we also considered a depth-first node-selection strategy. Such a strategy often quickly reaches an integer solution but may take longer to produce an overall optimal value.

In our experiments, we used a state-of-the-art MIP programming package called Cplex (Ilog, 2001a), which provides a set of libraries that allows one to customize the branch-and-bound search strategy. For example, one can vary node selection, variable-selection, variable setting strategies, or the LP solver. We used the default settings for the LP solver, which is primal simplex for the first node and dual simplex for subsequent nodes. We modified the search strategies to include some level of randomization. We randomized the variable-selection strategy by introducing noise into the ranking of the variables, based on maximum infeasibility. (Note that even in this scenario, the completeness of the search method is maintained.) We experimented with several other randomization strategies. For example, in Cplex one can assign an *a priori* variable ranking, which is fixed throughout the branch-and-bound process. We experimented with randomizing this *a priori* ranking, and we found that the dynamic randomized variable-selection strategy described above is more effective (Gomes and Selman, 1999).

In our experiments we were interested in problems that combine a hard combinatorial component with numerical information. Integrating numerical information into standard AI formalism is becoming of increasing importance. In planning, for example, one would like to incorporate resource constraints or a measure of overall plan quality. We considered examples that are based on logistics-planning problems but are formulated as mixed integer programming problems. These formulations extend the traditional AI planning approach by combining the hard constraints of the planning operators (the initial state and goal state) with a series of soft constraints that capture resource utilization. Such formulations have been shown to be very promising for modeling AI planning problems ( Kautz and Walser, 1999; Vossen et al., 1999).

Experimentation with the Cplex MIP solver showed that these problem instances are characterized by a non-trivial feasibility component.[4]

In figure 1.13(a), we compare the runtime profile of a depth-first strategy with that of a best-bound strategy to solve a hard feasibility problem in the logistics domain, formulated as a mixed integer programming problem. The

[4]We thank Henry Kautz and Joachim Walser for providing us with MIP formulations of the logistics-planning problems.

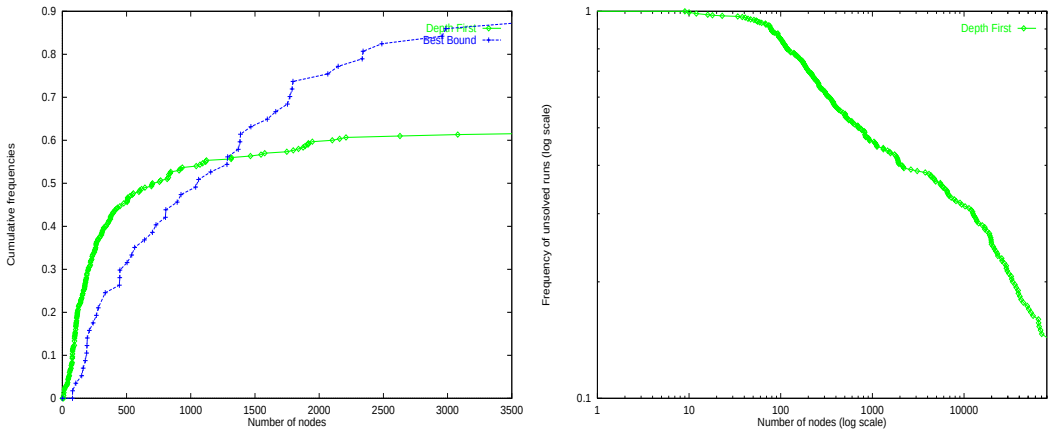


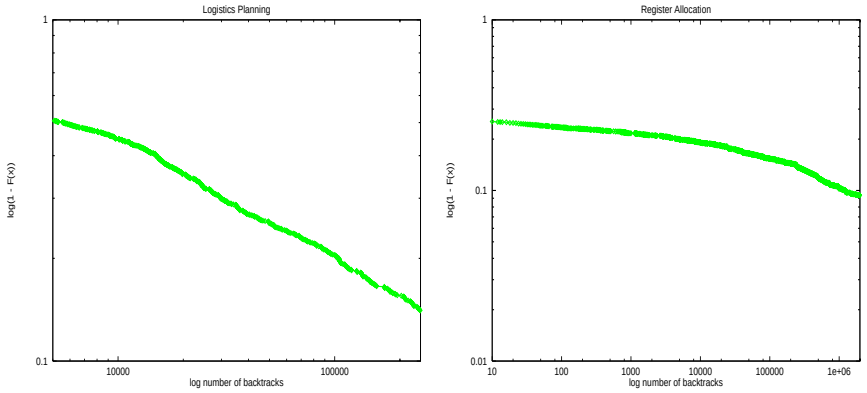

*Figure 1.13.* (a) Cost profiles for a logistics planning problems for depth-first and best-bound search strategies. (b) Hevay-tailed behavior of depth-first search.

search is terminated when an optimal or near-optimal ( within 10% of optimal) solution is found, but without the requirement of *proving* optimality. The figure shows the cumulative distribution of the solution time (in number of expanded nodes). For example, with 500 or fewer nodes, the depth-first search finds a solution on approximately 50% of the runs. Each run had a time limit of 5000 seconds. As we see from the figure, depth-first search initially outperforms best-bound search. After more than 1500 node expansions, however, the best-bound approach becomes more effective. For example, best-bound search finds a solution on approximately 75% of the runs with 2000 or fewer node expansions. In contrast, depth-first search could find a solution on only 55% of the runs with that many node expansions. These data are consistent with the observation above that the best-bound method may take more time to find an intial integer solution. Once such an initial integer solution is found, however, best-bound search becomes more effective.

We now look at the runtime distributions more closely. figure 1.13(b) gives a log–log plot of the complement-to-one of the cumulative distribution for the depth-first procedure. For example, from this plot we see that after 10,000 nodes, approximately 30% of the runs have not yet found a solution. The figure shows near-linear behavior over several orders of magnitude, an indication of heavy-tailedness. The curve for best-bound search also appears to exhibit heavy-tailed behavior, but less dramatically than that for depth-first search.

## 5.3 Boolean Satisfiability Formulations

We have also analyzed the runtime distributions of problems encoded as Boolean satisfiability (SAT) problems. The domains considered were AI planning, code optimization, and timetabling (Gomes et al., 1998a). We randomized




(a) (b)

*Figure 1.14.* Log-log plot of heavy-tailed behavior for SAT formulations of (a) logistics planning , and (b) register allocation

a complete SAT solver, Satz, by Li and Anbulagan, 1997. Satz is a version of the Davis–Putnam–Logemann-Loveland (DPLL) procedure (Davis and Putnam, 1960; Davis et al., 1979), with a highly tuned heuristic which is based on choosing a branching variable that, when set positively or negatively, maximizes a certain function of the number of unit propagations performed. Because Satz's heuristic usually chooses a single branching variable without ties, randomization of Satz was implemented using the uniform $H$-percent window rule (see section 2). Kautz and Selman, 1996, showed that propositional SAT encodings of difficult STRIPS-style AI planning problems could be efficiently solved by SAT engines. We considered the particular domain of "logistics planning," which involves moving packages on trucks and airplanes between different locations in different cities (Veloso, 1992).

A second set of problems involving SAT encodings comprised instances from the Dimacs Challenge benchmark (Johnson and Trick, 1996): a code-optimization problem, involving register allocation ("mulsol" instance), and circuit-synthesis problems ("adder" instances). For the circuit-synthesis problems, Kamath et al., 1990, developed a technique for expressing the problem of synthesizing a programmable logic array (PLA) as a SAT 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 becomes 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 were the "3bit-adder-32" and "3bit-adder-31", based on synthesizing a 3-bit adder using 32 and 31 gates, respectively.

Figure 1.14 displays log–log plots of the complement-to-one of the cumulative distributions for our SAT domains: the logistics-planning problem and the register-allocation problem. The approximately linear nature of the tails for several orders of magnitude indicates heavy-tailed behavior.

| Cases | $k$ | $\alpha$ |
|---|---|---|
| Logistics Planning (Satz) | 1,442 | 0.360 (0.017) |
| Register Allocation | 10,000 | 0.102 (0.002) |

*Table 1.3.* Estimates of the index of stability ($\alpha$). $k$ is the sample size. The values within parentheses are the estimated asymptotic standard deviations (SAT encoding).

Table 1.3 displays the maximum-likelihood estimates of the indices of stability (the values of $\alpha$) for the instances encoded as SAT problems. Note that for all the instances shown in the table, the estimates of $\alpha$ are consistent with the hypothesis of an infinite mean and an infinite variance, since $\alpha < 1$. We should point out, however, that given the computational difficulty of these problems, we had to truncate a larger portion of the tails of these distributions than we did in the previous examples, which renders our estimates of the indices of stability for the SAT-encoded problems less well-grounded in comparison (see also section 5.5).

## 5.4 Graph Coloring Formulations

In a study of the coloring of graphs with so-called small-world properties, Walsh, 1999,[5] also identifies heavy-tailed behavior. In a small-world topology, nodes are highly clustered, yet the path length between them is small (Walsh, 1999; Watts and Strogatz, 1998) . In contrast, random graphs tend to have short path lengths but little clustering, while regular graphs tend to have high clustering but large path lengths.

Walsh used random reordering of the variables to "randomize" a deterministic exact coloring algorithm based on DSATUR, by Trick, 1996 (see also section 2). Figure 1.15(a) shows a log–log plot of the tails of the runtime distributions of the complete backtrack search algorithm on three different graph

[5] We thank Toby Walsh for providing us with the plot for the study of coloring graphs with different structures.

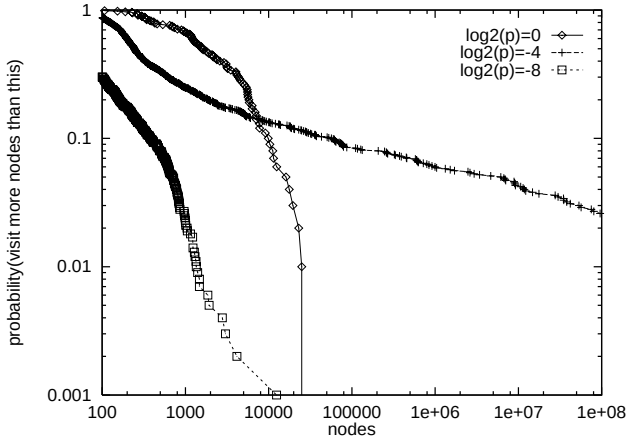



*Figure 1.15.* Log–log plot of the tail of the runtime distributions of an exact backtrack search coloring algorithm on three different graph topologies: random ($log_2(p) = 0$); small-world ($log_2(p) = -4$); and more structured ($log_2(p) = -8$). $p$ is the probability of randomly rewiring an edge of a regular graph. Size of the regular graph: 100 nodes and degree 8 ( Walsh, 1999).

structures: random, small-world, and a more structured topology. The approximately linear behavior of the tail of the distribution that corresponds to the small-world topology, *over 8 orders of magnitude*, is a clear indication of heavy-tailedness. (About 0.3% of the runs were truncated.) In sharp contrast to that case, the tails of the distributions corresponding to the random topology and the more structured topology have a very fast drop-off, indicating much lighter tails. Walsh conjectures that a small-world topology induces heavy-tailed behavior.

We also studied the runtime distributions of Dimacs color instances for school timetabling (Johnson and Trick, 1996). Figure 1.16 gives the tail of the distribution for the school timetabling instance, *school1_nsh.col*. The figure displays a log–log plot of the complement-to-one of the cumulative distribution. Again, the approximately linear nature of the tail over several orders of magnitude indicates heavy-tailed behavior. (About 3% of the runs were truncated.) The estimate for the index of stability ( $\alpha = 0.219$ ; see table 1.4) provides further evidence of the heavy-tailedness of the distribution.

## 5.5 Discussion

**Finite Search Space.**

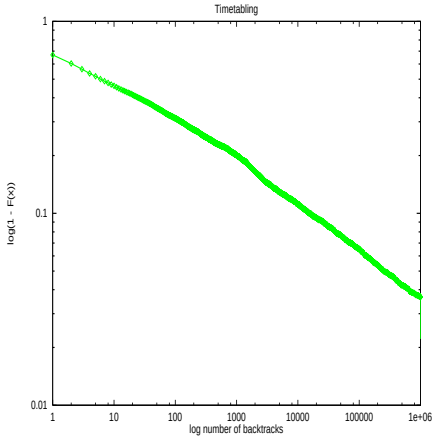



*Figure 1.16.* Log–log plot of heavy-tailed behavior for the timetabling problem

| Cases | $k$ | $\alpha$ |
|---|---|---|
| School Timetabling | 10000 | 0.219 (0.004) |

*Table 1.4.* Estimate of the index of stability ($\alpha$) for the school-timetabling problem. $k$ is the sample size. The value within parentheses is the estimated asymptotic standard deviation

In general, the computational cost of a complete backtrack algorithm has a finite upper bound.[6] Technically speaking, within the realm of finite bounds it is not rigorous to speak of distributions with infinite moments. Nevertheless, the heavy-tailed model provides a good approximation for the tail behavior of the search cost when dealing with the degree of complexity inherent in NP-complete problems. The heavy-tailed model emphasizes the persistence of power-law decay in the tail of the distribution over several orders of magnitude. Also, since the upper bound of the search space is exponential in the size of the problem, it is generally not reached in practice. As mentioned in section 3.2, one can also speak of *bounded* heavy-tailed behavior in distributions that exhibit power-law decay over an extensive portion of the tail but, given the finiteness of the search space, show a sudden drop-off further out in the tail. Thus, when allowing for a very large number of backtracks, so that one can explore most of the complete search tree, one will observe boundedness effects in the tail of the distribution.

[6]There are some applications that are characterized by unbounded search spaces —for example, in the area of theorem proving.

**Tail Truncation.**

As mentioned in section 4.1.3, in practice, we often have to impose a cutoff on the number of backtracks allowed, especially when studying the runtime distributions of hard instances. In other words, we have to *censor* the data, to avoid literally getting stuck in the most extreme outliers in the tail of the distribution during our data collection. As a consequence, the tail gets *truncated*. The selection of the *cutoff* depends on the problem instance and algorithm. Obviously, the higher the better. In general, we try to get a fairly large range, typically over at least 5 to 6 orders of magnitude in the value of the random variable, so that we can see some indication of stabilization in the tail behavior. Ideally, we would like to refrain from truncating a significant portion of the tail, but sometimes this is prohibitive from a computational stand point.

The more severe the truncation, the more difficult it becomes to differentiate between *heavy*-tailedness and *fat*-tailedness. From a practical point of view, this is not a problem, as long we have a clear indication that the tail exhibits behavior of one of these two types. Heavy-tailedness is not a necessary condition for speed-ups in randomized search methods. In fact, one can also speed up the performance of randomized backtrack search methods in the presence of *fat* tails. Nevertheless, the heavier the tail, the higher the frequency of very long runs — and as a result, the greater the advantage we stand to gain from randomization and restarts, so the more dramatic the speed-ups.

**Mixtures of Distributions.**

When modeling real-world phenomena, one often has to combine "mixtures" of distributions. For example, it is not uncommon to model income distributions with a mixture of a lognormal model, for the body of the distribution, and a Pareto model for the tail. One can also combine mixtures of the same family of distributions, *e.g.*, mixtures of exponential distributions. ( In fact, it has formally been proved that any distribution can be approximated by a mixture of exponential distributions.) In the study of computational phenomena, we often observe that the behavior of randomized backtrack search methods is best modeled by mixtures of distributions. For example, in figure 1.12 (round-robin sports scheduling problem) we can see that a mixture of distributions (for example, one for the body ($X \leq 10,000$) and another to model a Pareto-Lévy tail), would be a good approach to take. (See also Hoos, 2000 in which mixtures of exponential distributions are proposed to model certain runtime distributions.)

**Stronger Search Methods.**

Clearly, the existence and nature of heavy- or fat-tailed behavior depends largely on the particular randomized backtrack search algorithm used. Sometimes, just varying the randomization method causes the tails to change dramatically.

Let us consider the effect of more computation-intensive propagation methods on heavy-tailed behavior. In the experiemnts on QCP described earlier, for example, we used a simple backtrack search method with forward checking. When using stronger propagation techniques, such as general arc consistency, one can solve substantially larger problems (Régin, 1994; Shaw et al., 1998; Gomes et al., 2000). We maintain general arc consistency on the $N$-ary "all-different" constraints using the algorithm of Régin, 1994. This method uses a matching algorithm to ensure that, at each point during the search, in each row the set of remaining colors available for the yet-to-be-assigned cells is such that there is a different color available for each cell, and similarly for the columns. Using an efficient CSP-based implementation, we can solve instances of order 30 in under five minutes using uncensored data, which allows us to gather runtime information within a reasonable time frame.

By using general arc consistency for QCP, we observed that the tails became considerably *lighter* for quasigroups of low order. The reason for this is that those instances are solved mostly by propagation, with pratically no search. As we increased the order of the quasigroups, however, the heavy-tail phenomenon reappeared. The left panel in figure 1.17 shows the heavy tail for an instance of order 30 with 55% pre-assignment when applying the all-different constraints and using uncensored data (55% is close to the phase transition for order 30; Gomes et al., 2000.)

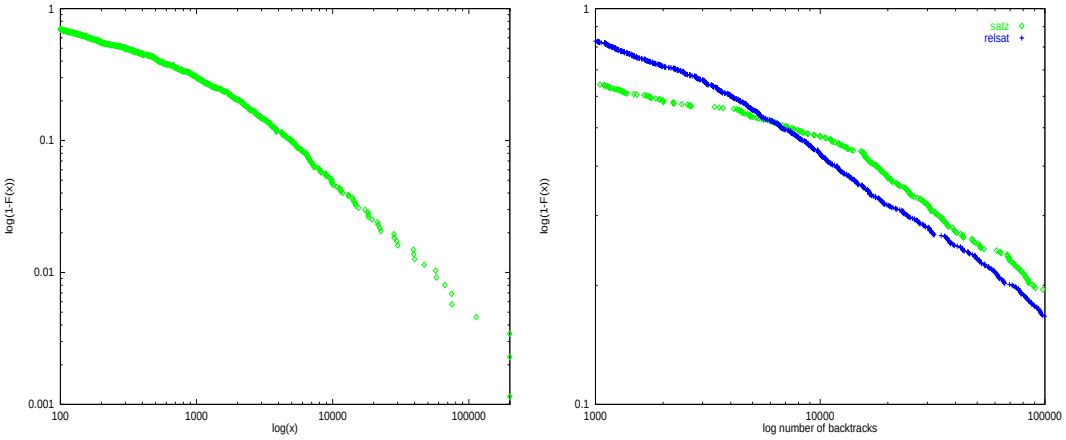



*Figure 1.17.* Left panel: heavy tail for quasigroup completion with extensive propagation (CSP formulation, order 30, with 55% pre-asssignment; uncensored data). Right panel: comparison of Satz to Relsat on logistics planning (SAT formulations).

| Cases | $k$ | $\alpha$ |
|---|---|---|
| Logistics Planning (Satz) | 1442 | 0.360 (0.017) |
| Logistics Planning (Relsat) | 1000 | 0.670 (0.009) |

*Table 1.5.* Index of stability: impact of look-back strategies on heavy-tailed behavior (SAT encodings).

The right panel in figure 1.17 shows a comparison between two different SAT procedures applied to the logistics-planning problem. We compare Satz (Li and Anbulagan, 1997) with Relsat (Bayardo and Schrag, 1997). As mentioned earlier, both procedures use sophisticated propagation rules and heuristics to guide their search. However, the Relsat backtracking strategy also includes a look-back technique, based on conflict-directed backjumping and relevance-bounded learning (Prosser, 1993; Bayardo and Miranker, 1996). We present this comparison to demonstrate the possible influence of such look-back strategies on heavy-tailed behavior. From the figure, we see that Relsat results in the *lighter* of the two tails, which is reflected in the steeper slope of the log–log plot. This means that the index of stability for Relsat is higher (see table 1.5). This observation is consistent with related work in which look-back strategies were shown to be effective in solving so-called exceptionally hard SAT instances (Bayardo and Schrag, 1997). In effect, the look-back approach reduces the degree of variability in the search, though it does not necessarily eliminate it altogether.

**Non–Heavy-Tailed Behavior.**

Clearly, the phenomenon of heavy-tailedness results from the interactions between the randomized algorithm and a particular instance. In fact, in general, for a given randomized backtrack-style algorithm and a problem domain, one should not expect to see heavy-tailed behavior on all instances. Typically, we observe the following pattern: On extremely easy instances, the tail of the runtime distribution decays very fast. As the level of difficulty of the instance increases, the tail becomes heavier and heavier and eventually goes through a region where heavy-tailedness is clearly in evidence. Beyond this heavy-tailed region, the instances become so hard for the given algorithm that the runtime distribution shifts to the right, and all the runs become very long and terminate after roughly the same number of steps. As a consequence, the variance of the runtime distribution decreases, the heavy-tailedness disappears, and the heavy tail is replaced by a fat tail. As the degree of difficulty of the instances increases even further, the variance of the runtime distribution continues to decrease.

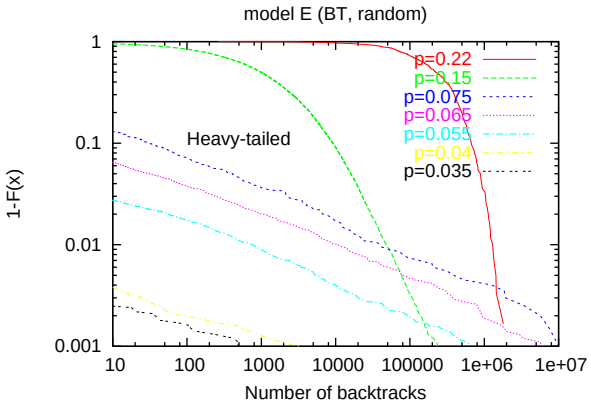



*Figure 1.18.* Survival function $(1-F(x))$ of the number of backtracks needed to solve different instances of model E with 20 variables and a domain size of 10. The parameter $p$ captures the constrainedness of the instances; it is the probability of choosing one constraint (out of the total number of possible constraints). Two different regimes in the decay of the survival function can be identified: a heavy-tailed regime (curves with linear behavior) and a non–heavy-tailed regime.

Figure 1.18 illustrates this phenomenon for a simple backtrack search algorithm (no look-ahead and no look-back) using a random variable-ordering heuristic in conjunction with random value selection. We studied random instances — generated using model E of random binary CSP (Achlioptas et al., 1997) — with different values of the constrainedess parameter $p$. In figure 1.18, the curves corresponding to instances with $p \leq 0.075$ exhibit Pareto-like behavior. The instances with $p$ right around 0.075 are clearly heavy tailed. To some extent, in fact, the boundary of this region constitutes a threshold for this particular backtrack search algorithm. Beyond this region, the instances become very hard and all the runs become homogeneously long, so that the variance of runtime distribution decreases and the tail of its "survival function" (which is just another name for the complement-to-one of the cumulative distribution) decays exponentially. For example, the curve corresponding to the instance with $p = 0.22$ exhibits exponential decay, which is much faster than linear decay.

Figure 1.19 shows log–log plots for two unsolvable instances from the quasigroup completion domain. One is a rare unsolvable instance in the underconstrained area (best fit: a gamma distribution); the other is in the critically constrained region (best fit: normal distribution). We see that both curves exhibit a sharp, rounded drop-off, which is indicative of the absence of heavy-tailed behavior.

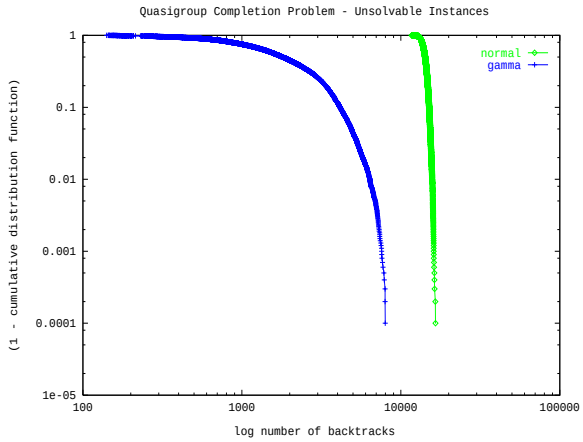



*Figure 1.19.* Absence of heavy-tailed behavior for unsolvable QCP instances (CSP formulation)

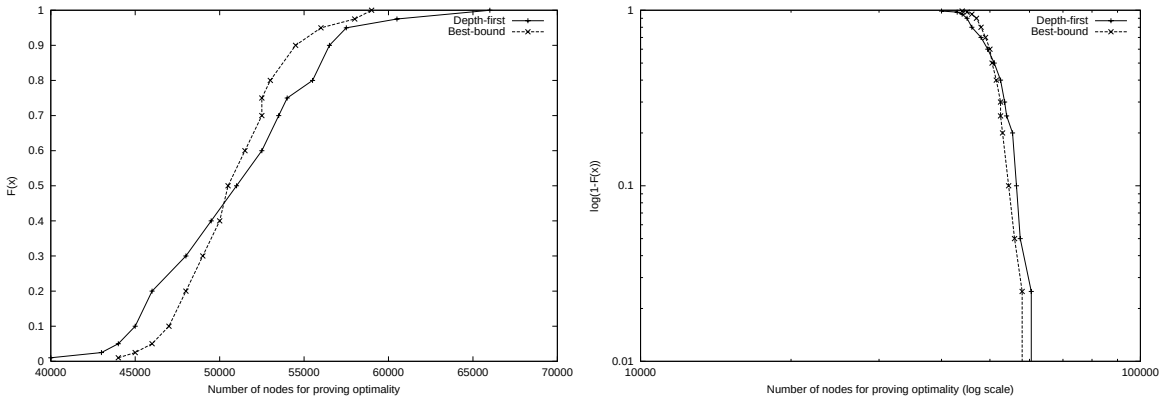



*Figure 1.20.* Left panel: comparison of runtime profiles for proving optimality of `misc07` from MIPLIB. Right panel: absence of heavy-tailed behavior in proving optimality of `misc07`

Figure 1.20 shows the distributions for proving optimality of the `misc07` problem from the MIPLIB library. It is apparent that the runtime for this instance exhibits relatively little variance in both the best-bound and depth-first searches. The sharp drop-off in the log–log plot in the left panel in the figure is a clear indication of the absence of heavy tails. Our experiments in proving optimality on other instances from the MIPLIB library also did not reveal heavy-tailed behavior.

We conjecture that methods for proving optimality and inconsistency may produce *fat* tails, but that it is less likely that they will produce *heavy*-tails, since the entire search space has to be explored and therefore the variance in the runtime would tend not to be very large. This is consistent with the

results by Frost et al., 1997, on proving inconsistency in constraint satisfaction problems. They showed that fat-tailed distributions, such as the Weibull and lognormal distributions, underlie the cost of proving inconsistency. Given the presence of fat tails in such cases, the use of randomization, in combination with restart strategies can still be effective in boosting backtrack search methods. In fact, as mentioned earlier, use of randomized strategies, together with restarts and nogood learning has been extremely successful in software and hardware verification, where unsatisfiability must be proved. Whether backtrack-style methods can produce heavy-tails in cases where optimality or inconsistency is to be proved is still an open question.

In the next section we show how the large variance in search methods, as characterized by heavy and fat-tailed behavior, can be exploited to boost the performance of randomized backtrack search methods by using restart and portfolio strategies.

## 6. Restart Strategies

We have shown that randomized backtrack search methods are characterized by *fat* (and, in some cases, even *heavy*) tails. The intuition is that, more often than one would expect, the heuristics guiding the search procedure make "wrong turns," causing the solver to be "stuck" in portions of the search space that contain no competitive solution — or no solution at all.

Given such a phenomenon, a randomized backtrack procedure is, in a sense, most effective early on in a search, which suggests that a sequence of short runs may be a more effective use of computational resources than a single long run. In this section and the next, we show how *restart* and *portfolio* strategies can boost performance of randomized search methods.

### 6.1 Elimination of Heavy-Tails

Figure 1.21(a) shows the result of applying a strategy of fixed-length short runs — *rapid randomized restarts* of a complete randomized search procedure — to a QCP instance of order 20 with 5% pre-assignment. The figure dislays a log–log plot of the tail of the distribution. From the figure, we see that without restarts and given a total of 50 backtracks, we have a failure rate of about 70%. Using restarts ( once after every 4 backtracks), this failure rate drops to about 10%. With an overall limit of only 150 backtracks, the restart strategy nearly always solves the instance, whereas the original procedure still has a failure rate of about 70%. Such a dramatic improvement due to restarts is typical for heavy-tailed distributions; in particular, we get similar results on critically constrained instances. The fact that the curve for the experiment with restarts takes a definite downward turn is a clear indication that the heavy-tailed nature of the original cost distribution has disappeared.

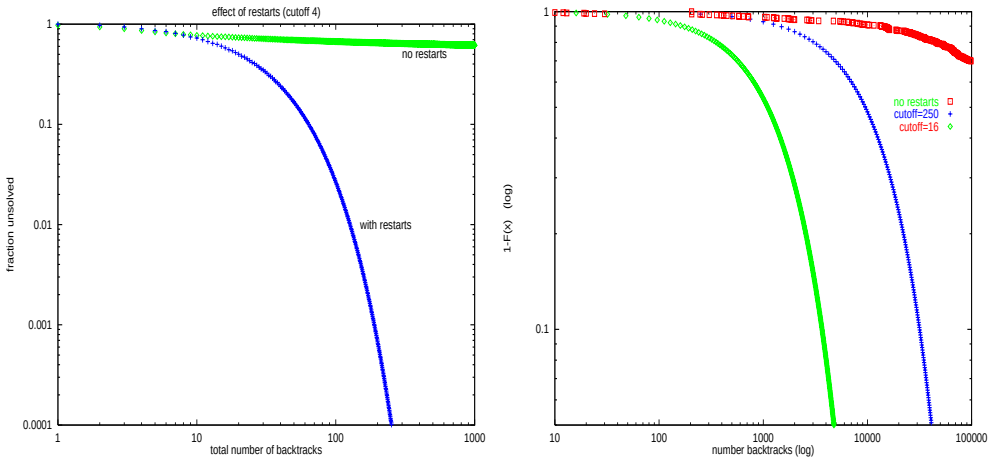



*Figure 1.21.* Restarts: (a) Tail (of $(1 - F(x))$ as a function of the total number of backtracks for a QCP instance (CSP formulation), log–log scale; (b) same as (a), but for a logistics instance (SAT formulation). In (b) the left-most curve is for a cutoff value of 16, the middle curve is for a cutoff of 250, and the right-most curve is without restarts.

Figure 1.21(b) shows the effect of restarts on the logistics-planning problem ( for the same instance as in figure 1.14(a)). The figure gives the distributions resulting from running randomized Satz with cutoffs (in the number of backtracks before each restart) of $16$ (near optimal) and $250$. The sharp drop-off in the log–log plot shows the disappearance of the heavy tail. We see, for example, that with restarts and using a cutoff value of 16, after a total of about 5,000 backtracks, we obtain a failure rate of approximately 5%. *Without restarts*, on the other hand, even after 100,000 backtracks the failure rate is still up around 70%.

The different plots in figure 1.21 clearly show that restarts shorten the tails of the distributions. Random restarts therefore provide an effective mechanism for dealing with *fat*- and *heavy*-tailed distributions. Of course, the cutoff parameter limits the size of the space that can be searched exhaustively between restarts. In practice, we gradually increase the cutoff, to allow us to maintain completeness (Gomes et al., 1998a).

We have proved formally that the underlying distribution of a restart strategy with a fixed cutoff eliminates heavy-tailed behavior, hence that all the moments of the distribution are finite. (The proof can be found in Gomes et al., 2000.)

## 6.2 Cutoff Value for Restart Strategy

| cutoff | success 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 |

(a)

| cutoff | success 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 |
| 128000 | 0.32 | 307,550 |

(b)

*Table 1.6.* Solving (a) a 16-team round-robin scheduling problem (CSP formulation) and (b) the logistics.d instance (SAT formulation) for a range of cutoff values.

Different cutoff values will result in different overall mean solution times. This can be seen in table 1.6, where mean solution times (based on 100 runs) for the round-robin scheduling problem (part (a), 16 teams; see also section 5.1) and the logistics-planning problem (part (b)) are considered for a range of cutoff values. The deterministic version of our CSP backtrack search was not able to solve the round robin for 16 teams.[7] The two sets of results in the table show the same overall pattern, revealing a clear optimal range of cutoff values. For the round-robin problem, the mean cost is minimized with a cutoff of about 50,000, and for the planning problem the optimal cutoff value is near 16. With a higher than optimal cutoff value, the heavy-tailed behavior to the right of the median begins to dominate the overall mean, whereas, for cutoffs below the optimal range the success rate is too low, requiring too many restarts to give a good overall value for the mean cost. These characteristics are apparent from table 1.6.

Figure 1.22 gives the data from table 1.6(b) in graphical form, with interpolation between data points. From the plot we see that the optimal cutoff value is around 12. The logarithmic vertical scale indicates that one can shift the performance of the procedure by several orders of magnitude by tuning the cutoff parameter.

In the previous examples, we used a *fixed cutoff* value $f$ for the restart strategy, which was calculated on the basis of the underlying empirical distribution. For the logistics-planning problem, the "optimal" cutoff value for the restart strategy is around 12, as illustrated in figure 1.22. In other words, we computed the fixed cutoff value $f$ that minimizes the expected runtime if the search procedure is

[7] CP techniques can now solve the round-robin sports-scheduling problem up to 40 teams (Regin, 2002)

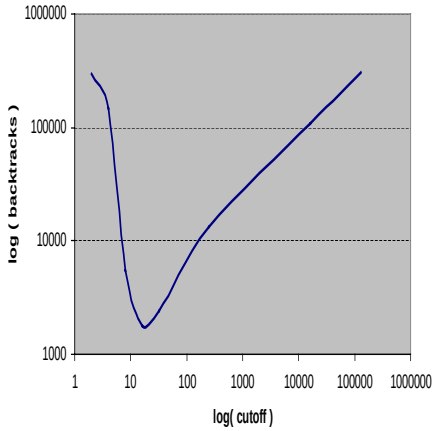



*Figure 1.22.* The effect of random restarts on solution cost for the logistics.d planning problem (SAT formulation)

| Cutoff | Num. Runs | Succ. Run Backtracks | Succ. Run Time |
|---|---|---|---|
| 50000 | 100 | | |
| | | 23420 | 629.89 |
| 250000 | 100 | | |
| | | 130613 | 990.8 |
| | | 165859 | 1034.39 |
| 500000 | 100 | | |
| | | 43620 | 653.52 |
| 1000000 | 100 | N.A. | N.A. |

*Table 1.7.* Restarts using a hybrid CSP/LP randomized backtrack search on a balanced QCP instance of order 50, at the phase transition.

restarted after every $f$ backtracks, until the solution is found, assuming that the runs are mutually independent.

Walsh, 1999, introduced a novel restart strategy, inspired by the analysis carried out by Luby *et al.*, in which the cutoff value increases geometrically, after each restart. The advantage of such a strategy is that it is less sensitive to the details of the underlying distribution.

As mentioned in the Introduction, state-of-the-art SAT solvers now incorporate restarts. In practice, the solvers use a default cutoff value, which is increased, linearly, after a specified number of restarts, guaranteeing the completeness of the solver in the limit (Moskewicz et al., 2001).

## 6.3 Formal Results on Restarts

The idea of a fixed-cutoff restart strategy is based on theoretical results by Luby et al., 1993, which describe provably optimal restart policies. In cases where the underlying runtime distribution of a randomized procedure is fully known, they showed that the optimal policy is the restart strategy with a fixed cutoff $f$, that minimizes the expected runtime. In our case, therefore, where we assume that the empirically determined distribution of our search procedure is a good approximation of the real distribution, the "optimal" strategy is just a sequence of runs of fixed length.

Luby *et al.* also provide a strategy for minimizing the expected cost of randomized procedures even in cases where there is no *a priori* knowledge about the distribution—a *universal* strategy that is suitable for all distributions. The universal strategy consists of a sequence of runs each of whose lengths is some power of two. Every time a pair of runs of a given length has been completed, a run of twice that length is immediately executed. The universal strategy is of the form: $1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 4, 8 \cdots$.

Although the universal strategy of Luby *et al.* is provably within a constant log factor (in the *limit*) of the strategy that results from using a fixed cutoff (the optimal strategy to be used when the distribution is known), we found that in practice the schedule often converges too slowly. The work behind Luby's universal strategies was motivated by Ertel's observation of potentially long runs of theorem proving methods and their effect on parallel strategies (Ertel and Luby, 1994).

In Williams et al., 2003a; Williams et al., 2003b the connections between so-called backdoors, restarts, and the phenomenon of heavy-tailedness are described. Backdoors are small sets of variables that capture the overall combinatorics of a problem instance. More precisely, a backdoor is a set of variables for which there is a value assignment such that the simplified problem can be solved by a poly-time algorithm, called the "subsolver." The subsolver captures any form of poly-time simplification procedure as used in current CSP solvers. Of course, the set of all variables in a problem forms a trivial backdoor set, but many interesting practical problem instances possess much smaller backdoors and strong backdoors. When given a set of backdoor variables of a problem instance, one can restrict the combinatorial search by branching only on the backdoor variables and thus search a drastically reduced space. Interestingly, Williams et al., 2003a observed that structured problem instances can have suprisingly small sets of backdoor variables.

The notion of backdoor came about in the context of the study of heavy-tailed behavior as observed in backtrack-style search. Heavy-tailed distributions provide a justification of why restarts are effective, namely to prevent the search

procedure from getting stuck in unproductive 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. This is where backdoors enter the picture: Intuitively, a small backdoor explains how a backtrack search can get "lucky" on certain runs, since the backdoor variables are identified early on in the search and set in the right way.

Williams et al., 2003a; Williams et al., 2003b provide a detailed mathematical model that explains heavy-tailed behavior (Pareto-like tails) in backtrack search as a function of the size of a minimal backdoor set. They show, furthermore, that even though finding a small set of backdoor variables is computationally hard, the very existence of a small backdoor in a problem provides a concrete computational advantage to its solution. They consider three scenarios. First, a deterministic scenario is considered with an exhaustive search of backdoor sets. In this scenario, one obtains provably better search complexity when the backdoor contains up to a certain fraction of all variables. They then show that a randomized search technique, which in effect repeatedly guesses backdoor sets, provably outperforms a deterministic search. Finally, in the third scenario the availability of a variable selection heuristic is considered, which provides guidance in looking for a backdoor set. This strategy can reduce the search space even further. By exploiting restart strategies, for example, one can obtain a polynomially solvable case when the backdoor contains at most $\log(n)$ variables, where $n$ is the total number of variables in the problem instance. We believe that this final scenario is closest to the behavior of the effective randomized backtrack SAT and constraint solvers that are currently available.

## 6.4 Restart Results on a Range of Problem Instances

Table1.7 shows the results of randomization and restarts on a *balanced* QCP instance of size 50, and right at the *phase transition*—a very hard instance, not previously solved by any other complete search method. A balanced instance is one in which the numbers of pre-assigned cells in the various rows and columns are nearly equal to one another. Balanced QCP instances are considerably harder than instances in which the numbers of pre-assigned cells are randomly distributed (Kautz et al., 2001). We used a complete backtrack search method that combines CSP propagation techniques with linear programming (LP) randomized rounding (Gomes and Shmoys, 2002). The LP randomized rounding is based on the relaxation of a 0-1 integer programming formulation of the QCP problem. In the LP randomized rounding the optimal variable values may be fractional, between 0 and 1. The optimal value of each such variable is interpreted as the probability that the variable will be assigned the value 1. The LP randomized rounding is used within the backtrack procedure as a heuristic

for selecting the next variable/value to be assigned. (See Gomes and Shmoys, 2002, for details.)

From the table we can see that we could only solve the instance in only one out of 100 runs when using a cutoff of $50,000$ backtracks, in only out of 100, when using a cutoff of $250,000$ backtracks, and in only one out of 100, when using a cutoff of $500,000$ backtracks. We also ran the backtrack search procedure with a cutoff of $1,000,000$ backtracks, but with such a high cutoff we were not able to solve the instance at all in 100 runs. In general, complete backtrack search methods that incorporate randomization and restarts allow us to solve considerably harder QCP instances than the standard, deterministc backtrack procedures.

| Problem | Solver | Deterministic soln. time | RRR 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 |
| blocks-world.d | Satz | 30 min | 23 min |

*Table 1.8.* Randomized rapid restarts (RRR) versus deterministic versions of backtrack search procedures (Satz solver used on SAT encodings; Ilog solver on CSP encodings).

In table 1.8 we give the mean solution times of randomized backtrack procedures with rapid restarts (RRR), for a range of problem instances. In each case, the runtimes were averaged over 100 runs. For comparison, we include the runtime of the original deterministic procedure (SAT for the logistics, adder, and blocks-world instances; CSP for the round-robin problems).[8]

Our deterministic CSP procedure on the round-robin scheduling problem gives us a solution for the 14 team problem in 411 seconds; randomization improves this to 250 seconds. We could not find solutions for the 16 and 18 team problem with the deterministic version. Apparently, the subtle interaction between global and local constraints makes the search for a globally consistent solution surprisingly difficult. These problem instances are too hard to obtain a full cost distribution, making it difficult to calculate an optimal cutoff value.[9]

[8]The deterministic runs can also be viewed as single runs of the randomized procedure with an infinite cutoff value. Note that, of course, one might be "lucky" on any given instance and have the deterministic procedure branch in just the right way. On the harder instances, however, we often need several hundred restarts to find a solution. On those instances, it becomes quite unlikely that a single deterministic run would succeed.

[9]For problems for which we can empirically determine the overall cost profile, we can use the empirical distribution to calculate the cutoff value that minimizes the expected cost of finding a solution.

For example, in the 16 team case, running with a cutoff of 1,000,000 backtracks gives a success rate of less than 40%, so, we do not even reach the median point of the distribution. (We estimate that the median value is around 2,000,000 backtracks.) Each run takes about 2 hours to complete.

In order to find a good cutoff value for very hard problem instances, one possible strategy is to start at relatively low cutoff values, since the optimal cutoff for these problems tends to lie far below the median value of the distribution, and incrementally increase the cutoff in order to guarantee the completeness of the algorithm. Using such a strategy with RRR, we were able to solve the 16 team instance in 1.4 hours and the 18 team in approximately 22 hours.

For the SAT encodings of the 3bit-adder problems, which are examples of Boolean-circuit synthesis problems from the Dimacs benchmark ( Johnson and Trick, 1996), the RRR solution times are—to the best of our knowledge—the first successful runs of a backtrack search procedure (DPLL) on these instances, though they were previously solved with local search methods, (Selman and Kautz, 1993).

The results given in tables 1.7 and 1.8 show that introducing a stochastic element into a backtrack-style SAT or CSP 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 several previously unsolved problem instances.

## 6.5 Learning Dynamic Restart Strategies

In section 6.2 we described a fixed-cutoff restart strategy based on the work of Luby *et al.*. Though these results are quite important, it is often mistakenly assumed that they solved the issue of restart strategies *completely*. It should be noted that the analysis of Luby *et al.* of their universal strategy is based on a few key assumptions, namely that (1) no information about the prior run-time distribution of the solver on the given instance, or complete information (2) the only information we have about the solver is the runtime; (3) we have unlimited time and resources to solve the problem.

In general, these assumptions turn out to be invalid when it comes to real-world scenarios—in fact, quite often we have *partial* knowledge about the runtime distribution of a given instance. For example, we know that the median run time for solving a QCP instance in the under-constrained area is considerably shorter than for an instance at or near the phase transition. It can be shown that if one has prior knowledge about the problem instance (such as experience in solving instances of the same problem class), or if one has some time constraints (*e.g.*, a need to solve the instance within, say, two days of cpu time), then we can improve upon the fixed-cutoff restart strategy.

Motivated by these considerations, Horvitz et al., 2001, introduced a Bayesian framework for learning predictive models of randomized backtrack solvers. The basic idea is to predict the length of a run based on several features of the runs, captured during the initial execution of the runs. The features considered were both domain dependent and domain independent, such as: depth of backtrack points, number of unit propagations, size of the remaining search tree.

Extending that work, Kautz et al., 2002, considered restart policies that can factor in information about a solver's behavior which is based on real-time observations. In particular, they introduce an *optimal* policy for dynamic restarts that takes such observations into account. To demonstrate the efficacy of their restart policy, they carried out empirical studies of two randomized backtrack solvers, one for CSP formulations and the other for SAT formulations. Ruan et al., 2002, further extended this work by considering the case of dependency among runs, when the runs have the same underlying distribution, given two different *a priori* distributions. They show how, in such a scenario, an offline dynamic programming approach can be used to generate the optimal restart strategy, and how to combine the resulting policy with real-time observations, in order to boost the performance of backtrack search methods.

The study of dynamic restart strategies based on learning models about the behavior of the solvers is a new are of research, offering a new class of procedures for tackling hard computational problems. Clearly, much research is needed in this new emerging area.

## 6.6 Variants of Restart Strategies

Lynce et al., 2001, propose "randomized backtracking", a technique for randomizing the *target* decision points of a backtrack solver, *i.e.*, the points to which the solver backtracks during the search. Such randomized target points are decided based on randomly picking a decision variable from learned conflict clauses. This backtracking strategy contrasts with the standard, non-chronological backtracking strategy, in which the most recent decision variable is selected as the target backtrack point.

Several variants of this "randomized backtracking" protocol can be considered, namely a *destructive* strategy that undoes all the decisions performed after the *target* decision point, or a *non-destructive* strategy that undoes only the decision associated with the target point itself, leaving all the other assignments unaffected. The former strategy is more drastic, since it basically erases a portion of the search tree previously searched. Non-destructive backtracking has the flavor of a local search procedure, since the current assignment is only locally modified.

Lynce *et al.* propose a parameter $K$ that controls the frequency with which to apply randomized backtracking. When $K$ is set to 1, "randomized back-

tracking" is performed every time a conflict is detected; otherwise, randomized backtracking is performed every $K$ conflicts. In the latter case, either just the most recent conflict or all the interim conflicts can be considered when picking the *target* point.

Both *destructive* and *non-destructive* procedures can lead to *unstable* algorithms. Lynce *et al.* discuss ways of dealing with such situations in order to guarantee the completeness of the algorithm (*e.g.*, by preventing clause deletion). Even though the number of recorded clauses can grow exponentially in the worst-case, in practice (based on experiments with real-world instances) this does not seem to pose a problem (Lynce et al., 2001).

An alternative to "randomized backtracking," proposed by Lynce *et al.* is *unrestricted backtracking*, a form of "deterministic randomization", that uses a deterministic formula (specifically, a function of the clauses learned during search) to select the target point. This strategy is guaranteed to be complete, and the system will still behave (deterministically) randomly (*i.e.*, it is very unlikely that it will pick a *target point* previously selected). Analogously to randomized backtracking, unrestricted backtracking can also be applied every $K$ conflicts.

Zhang, 2002, introduces a "random jump" strategy, which can be viewed as a variant of the complete restart strategy. Like restarts, the main motivation behind the strategy proposed by Zhang is to allow the solver to escape unproductive portions of the search tree. However, rather than restarting from scratch, the backtrack solver randomly jumps to unexplored portions of the search space, ensuring that no portion of the search space is visited twice. The jumping decisions are based on analyzing, at given *checkpoints*, to what extent the remaining search space is too large to be fully explored in the remaining time allotted for the search. If the percentage of the remaining space is larger than that of the remaining allotted time, the remaining space is considered *sufficiently large*, and so the solver skips some open right branches of the search tree along the way. The *checkpoints* of Zhang's strategy are analogous to the cutoffs for the fixed restart strategy

Zhang's "random jump" strategy is implemented in SATO, an efficient implementation of the DPLL backtrack search procedure for Boolean satisfiability. The strategy is quite effective, and it has solved a dozen previously open problems in the area of finite algebra. As reported in Zhang, 2002, SATO, could not solve those open problems considered— even with a full week of runtime allotted per problem—by using its standard approach. With the "random jump" strategy, SATO was able to solve each of them in an overnight run.

As a final point, we mention "probing." Probing is extensively used in CSP and MIP formulations (see *e.g.*, Crawford and Baker, 1994 and Savelsbergh, 1994). In a "probing" strategy one repeatedly goes down a random branch of

the search tree, with no backtracking, which can be seen as a restart with a cutoff of 1.

In the next section we discuss portfolio strategies, another technique for combatting the high variance of search methods.

## 7. Portfolio Strategies

In the previous sections we showed how the performance of backtrack search methods can vary significantly from instance to instance—and, when using different random seeds, even on the same instance. One can take advantage of such high variance by combining several algorithms into a portfolio, and either running them in parallel or interleaving them on a single processor. In this section we describe portfolio strategies and identify conditions under which they can have a dramatic computational advantage over the best traditional methods.

### 7.1 Portfolio Design

A *portfolio of algorithms* is a collection of different algorithms and/or different copies of the same algorithm running on different processors (Gomes and Selman, 1997b; Huberman et al., 1993).

We consider randomized algorithms—Las Vegas and Monte Carlo algorithms. Therefore, the computational cost associated with a portfolio is a random variable. The expected computational cost of the portfolio is simply the expected value of the random variable associated with the portfolio, and its standard deviation is a measure of the "dispersion" of the computational cost obtained when using the portfolio of algorithms. In this sense, the standard deviation is a measure of the risk inherent in using the portfolio.

The main motivation to combine different algorithms into a portfolio is to improve on the performance of the component algorithms, mainly in terms of expected computational cost but also in terms of the overall risk. As we will show, some portfolios are strictly preferable to others, in the sense that they provide a lower risk *and* a lower expected computational cost. However, in some cases, we cannot identify any portfolio (within a set that of portfolios) that is best, in terms of both expected value and risk. Such a set of portfolios corresponds to the *efficient set* or *efficient frontier*, following terminology used in the theory of mathematical finance (Martin et al., 1988). Within an efficient set, one has to accept deterioration in the expected value in order to minimize the risk; conversely, one has to assume greater risk in order to improve the expected value of the portfolio.

In this context, where we characterize a portfolio in terms of its mean and variance, combining different algorithms into a portfolio only makes sense if they exhibit different probability profiles and none of them *dominates* the others

over the whole spectrum of problem instances. An algorithm *A* dominates algorithm *B* if the cumulative frequency distribution function for algorithm *A* lies above the corresponding value for algorithm *B* at every point, that is for every value of the random varaible.

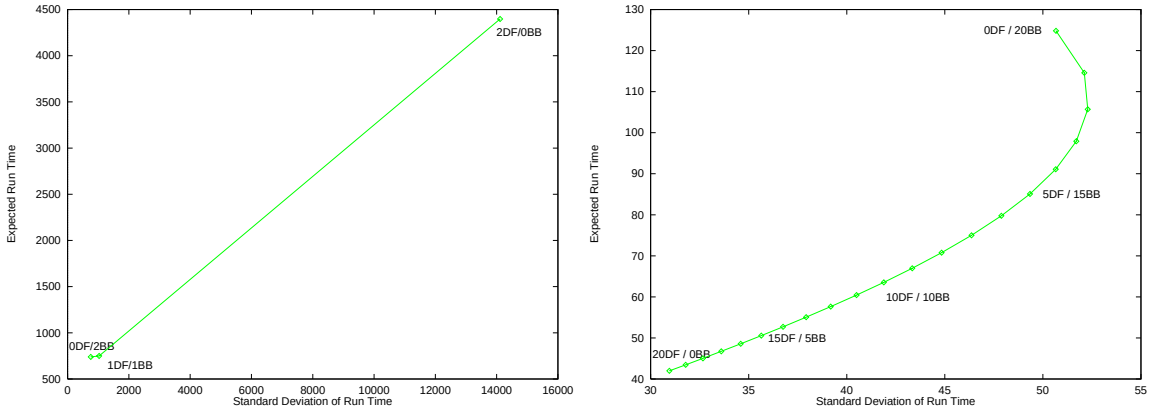



*Figure 1.23.* Portfolio results for logistics planning: left panel, 2 processors; right panel, 20 processors

## 7.2 Portfolio Results

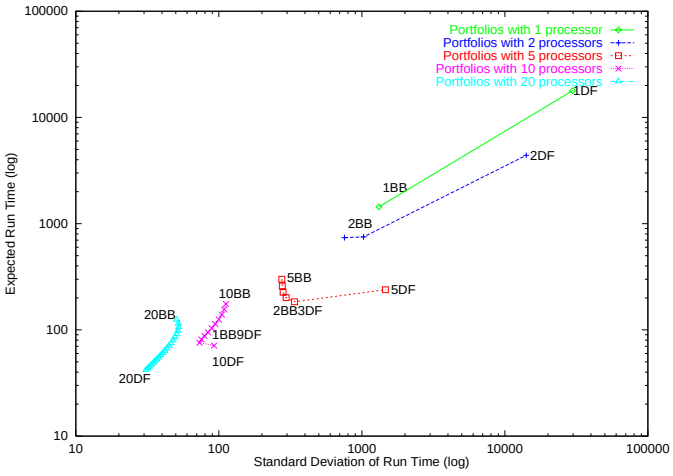



*Figure 1.24.* A range of portfolios for the MIP formulation of logistics planning (expected runtime).

In section 5.2 we have shown that there are several interesting trade-offs between depth-first branch-and-bound and best-bound branch-and-bound. In

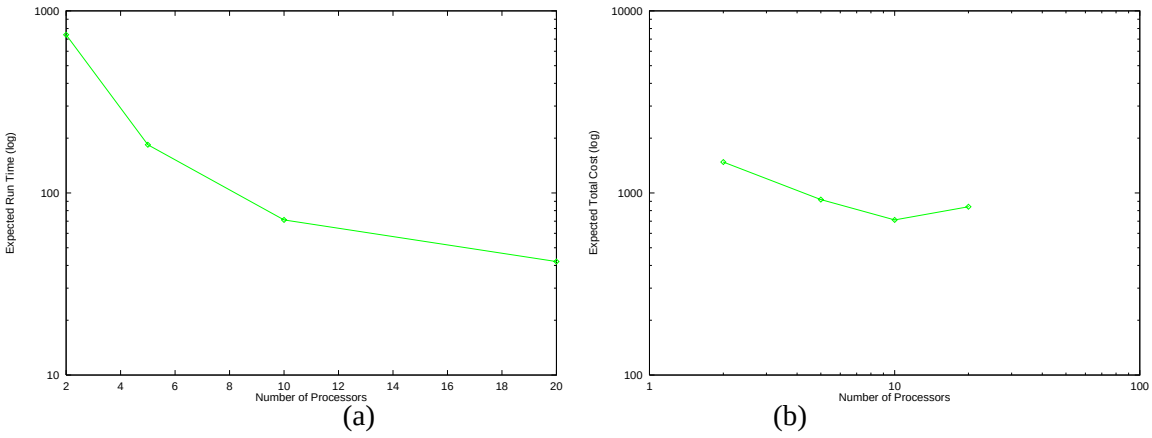



*Figure 1.25.* Expected runtime (a) and expected total cost (b) of optimal portfolio for different numbers of processors.

particular, depth-first search performs better early on in the search, whereas best-bound is better on longer runs. We also identified *fat* and *heavy*-tailed behavior in the cost profiles of the MIP algorithms.

These results indicate that choosing a single search strategy — which is the standard current practice — may not be the most effective approach when dealing with these types of MIP problems. In fact, as we will show below, one can obtain better results by combining strategies, either by interleaving them or by running them in parallel.

In this section we illustrate the applicability of the portfolio framework by using the results from section 5.2 for the general paradigm of mixed integer programming. The portfolio approach is general and can therefore be applied to exploit different types of algorithms, including mixtures of complete backtrack-style algorithms and local search algorithms.

In the left panel in figure 1.23, we present our results for the case of using two processors to solve the logistics problem (the same instance as the one to which figure 1.13 applies). The plot gives the expected runtime and standard deviation for different ways of combining a branch-and-bound search procedure using depth-first search and a branch-and-bound search procedure using best-bound search, assuming two processors. These values can be computed analytically from the cost profiles in figure 1.13, by using the portfolio probability distribution (see Gomes and Selman, 2001 for details). From the left panel in figure 1.23 we see that in the case of two processors, the best choice in terms of minimizing the expected runtime and standard deviation consists of running branch-and-bound with best-bound on both processors (i.e., 0DF/2BB). The expected runtime of such a strategy is approximately 700 nodes, with a stan-

dard deviation of about 750. Note the contrast between these values, and the much higher values corresponding to the strategy of running branch-and-bound with depth-first on both processors (i.e., 2DF/0BB, with an expected runtime of 4397 and a standard deviation 14,112).

From the right panel in figure 1.23 we see that in the case of 20 processors, the best strategy is to use only depth-first search (*i.e.*, 20DF/0BB). Note that this is quite counterintuitive, especially in the view that is exactly the opposite situation from that of the one- or two- processor scenario, where using only best-bound strategies is optimal. So, we see that in certain cases, *optimal behavior emerges from running many copies of a single suboptimal strategy.*

In figure 1.24, we present data for a range of portfolios that we used in solving our logistics problem.. Each of these portfolios consists of anywhere from one to twenty processors.

The figure gives the expected runtime and standard deviation for different ways of combining branch-and-bound search procedures using depth-first search with those that use best-bound search. From this plot we see that the mixing strategy changes as we increase the number of processors or the amount of interleaving. In the case of one processor, the best choice in terms of minimizing the expected runtime and standard deviation consists of running branch-and-bound with best-bound. Best-bound remains the best strategy when we scale up two processors (*i.e.*, the best portfolio consists of running branch-and-bound with best-bound on both processors, as mentioned earlier). In the experiments with five and ten processors, the best strategy is a combination of depth-first search and best-bound search. In the case of ten processors, for example, the best strategies are 9DF/1BB and 10DF/0BB. Neither of those two portfolios dominates; taken together, therefore, they constitute an efficient set. There is a tradeoff between the expected runtime and the standard deviation : The 10DF/0BB portfolio minimizes the expected runtime, while the 9DF/1BB portfolio minimizes the risk (standard deviation).

Figure 1.25(a) shows that once the number of processors exceeds 10, the expected runtime decreases at a very slow rate. In figure 1.25(b), we show the expected total cost of solving the chosen problem instance. It is clear from this figure that when total resource usage is factored in the most cost-effective solution[10] is obtained with a 10-processor portfolio.

## 7.3 Variants of Portfolios

Baptista and Marques-Silva, 2000, consider algorithm portfolios based on restarts: Each time the search is restarted, a *different* algorithm is selected

[10]The expected the total cost is the product of the time to find a solution and the number of processors in the portfolio.

from a set of algorithms, $\{A_1, A_2, \ldots, A_k\}$. Each algorithm $A_i$ has probably $p_i$ of being selected. A key aspect of their approach is that restarts are not independently performed: A database of learned clauses is maintained from run to run, allowing the system to reuse information about the search space learned during each run. In their approach, they select different configurations of the same algorithm (GRASP) for the portfolio mix, rather than using radically different algorithms; they vary parameters such as the branching heuristics and the amount of randomization. Baptista and Marques-Silva report very promising results with their portfolio approach: They were able to prove, for example, that several hard superscalar-processor verification instances were *unsatisfiable*.

Davenport, 2000, uses portfolios of a cooperative problem solving team of heuristics that evolve algorithms for a given problem instance. They solve difficult instances of a bicriteria sparse multiple knapsack problem based on real-world inventory instances.

Leyton-Brown et al., 2002 proposed a promising methodology for building portfolio of algorithms. Their approach uses machine learning techniques to identify key features of instances with respect to different algorithms, and was initially motivated by the combinatorial auction winner determination problem.

## 8. Conclusions

Researchers have informally observed that the performance of *complete* backtrack search methods can vary considerably from instance to instance. Early formal results showed that randomization and restarts can increase the robustness of such search methods. Nevertheless, until recently, such techniques were not believed to provide significant *practical* benefits in a *complete* search setting: Up until about five years ago, state-of-the-art complete backtrack search solvers did not exploit randomization and restart strategies.

Advances in the understanding of heavy-tailedness of certain runtime distributions of backtrack search methods has led to a dramatically different attitude toward randomization, restart, and portfolio strategies, for *complete* or *exact* methods. Restarts and portfolios can significantly reduce the variance in runtime and the probability of failure of such search procedures, resulting in search methods that are both more robust and more efficient overall. Randomization and restart strategies are now becoming an integral part of state-of-the-art complete backtrack solvers. We hope that this overview of the study and design of randomized complete backtrack search methods will stimulate additional research efforts in this new and exciting research area.

## Acknowledgments

I thank all my colleagues with whom I have collaborated and exchanged ideas on the study and design of randomized complete search methods over the past five years. In particular, I would like to thank Dimitris Achlioptas, Ramon Béjar, Christian Bessière, Hubie Chen, Nuno Crato, César Fernández, Eric Horvitz, Henry Kautz, Felip Manya, João Marques-Silva, Ken McAloon, Bart Selman, David Shmoys, Carol Tretkoff, Toby Walsh, Steve Wicker, and Ryan Williams. I thank Toby Walsh for providing me with the plot in figure 1.15, and Henry Kautz and Joachim Walser for the MIP formulations of the logistics-planning problems.

This research was partially supported by AFRL grants F30602-99-1-0005 and F30602-99-1-0006, AFOSR grant F49620-01-1-0076 (Intelligent Information Systems Institute) and F49620-01-1-0361 (MURI grant on Cooperative Control of Distributed Autonomous Vehicles in Adversarial Environments), and DARPA F30602-00-2-0530 (Controlling Computational Cost: Structure, Phase Transitions, and Randomization) and F30602-00-2-0558 (Configuring Wireless Transmission and Decentralized Data Processing for Generic Sensor Networks). The views and conclusions contained herein are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

## References

Achlioptas, D., Kirousis, L. M., Kranakis, E., Krizanc, D., Molloy, M. S. O., and Stamatiou, Y. C. (1997). Random constraint satisfaction: A more accurate picture. In *Principles and Practice of Constraint Programming*, pages 107–120.

Adler, R., Feldman, R., and Taqqu, M. (1998). *A practical guide to heavy tails*. Birkhäuser.

Agrawal, M., Kayal, N., and Saxena, N. (2002). Primes in P. (www.cse.iitk.ac.in/news/primality.html).

Baptista, L. and Marques-Silva, J. (2000). Using randomization and learning to solve hard real-world instances of satisfiability. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP00)*.

Bayardo, R. and Miranker, D. (1996). A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 558–562, Portland, OR. AAAI Press.

Bayardo, R. and Schrag, R. (1997). Using csp look-back techniques to solve real-world sat instances. In *Proceedings of the Fourteenth National Confer-*

*ence on Artificial Intelligence (AAAI-97)*, pages 203–208, New Providence, RI. AAAI Press.

Beasley, J. E. (1990). OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072.

Brelaz, D. (1979). New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256.

Chen, H., Gomes, C., and Selman, B. (2001). 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.

Colbourn, C. (1984). The complexity of completing partial latin squares. *Discrete Applied Mathematics*, (8):25–30.

Cook, S. and Mitchell, D. (1998). Finding hard instances of the satisfiability problem: a survey. In Du, Gu, and Pardalos, editors, *Satisfiability Problem: Theory and Applications. Dimacs Series in Discrete Mathematics and Theoretical Computer Science, Vol. 35*.

Crawford, J. and Baker, A. (1994). Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In *Proceedings of The Tenth National Conference on Artificial Intelligence*.

Davenport, A. (2000). Cooperative strategies for solving bicriteria sparse multiple knapsack problems. In *Proceedings of the AAAI Workshop on Leveraging Probability and Uncertainty in Computation*.

Davis, M., Logemann, G., and Loveland, D. (1979). A machine program for theorem proving. *Communications of the ACM*, 5:394–397.

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

Easton, K., Nemhauser, G., and Trick, M. (2001). The Traveling Tournament Problem: Description and Benchmarks. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming (CP01)*, Cyprus.

Ertel, W. and Luby, M. (1994). Optimal Parallelization of Las Vegas Algorithms. In *Symp. on Theoretical Aspects of Computer Science, Lect. Notes in Computer Science 775*, pages 463–475. Springer Verlag.

Feller, W. (1968). *An introduction to probability theory and its applications Vol. I*. John Wiley & Sons, New York.

Feller, W. (1971). *An introduction to probability theory and its applications Vol. II*. John Wiley & Sons, New York.

Frost, D., Rish, I., and Vila, L. (1997). Summarizing CSP hardness with continuous probability distributions. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, New Providence, RI. AAAI Press.

Gent, I. and Walsh, T. (1993). Easy Problems are Sometimes Hard. *Artificial Intelligence*, 70:335–345.

Goldberg, E. and Novikov, Y. (2002). Berkmin: A fast and robust sat solver. In *Proc. of Design Automation and Test in Europe (DATE-2002)*, pages 142–149.

Goldwasser, S. and Kilian, J. (1986). Almost all primes can be quickly certified. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 316–329.

Gomes, C. and Selman, B. (2001). Algorithm portfolios. *Artificial Intelligence*, 126:43–62.

Gomes, C. P. and Selman, B. (1997a). Algorithm Portfolio Design: Theory vs. Practice. In *Proceedings of the Thirteenth Conference On Uncertainty in Artificial Intelligence (UAI-97)*, Linz, Austria. Morgan Kaufman.

Gomes, C. P. and Selman, B. (1997b). Problem Structure in the Presence of Perturbations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, New Providence, RI. AAAI Press.

Gomes, C. P. and Selman, B. (1999). Search Strategies for Hybrid Search Spaces. In *Proceedings of the Eleventh International Conference on Tools with Artificial Intelligence (ICTAI-99)*, Chicago, IL. IEEE.

Gomes, C. P., Selman, B., and Crato, N. (1997). Heavy-tailed Distributions in Combinatorial Search. In *Proceedings of the Third International Conference of Constraint Programming (CP-97)*, Linz, Austria. Springer-Verlag.

Gomes, C. P., Selman, B., Crato, N., and Kautz, H. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1–2):67–100.

Gomes, C. P., Selman, B., and Kautz, H. (1998a). Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, New Providence, RI. AAAI Press.

Gomes, C. P., Selman, B., McAloon, K., and Tretkoff, C. (1998b). Randomization in Backtrack Search: Exploiting Heavy-Tailed Profiles for Solving Hard Scheduling Problems. In *Proceedings of Fourth International Conference on Artificial Intelligence Planning Systems 1998*, Pittsburgh, PA, USA.

Gomes, C. P. and Shmoys, D. (2002). The promise of LP to boost CSP techniques for combinatorial problems. In Jussien, N. and Laburthe, F., editors, *Proceedings of the Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR'02)*, pages 291–305, Le Croisic, France.

Hall, P. (1982). On some simple estimates of an exponent of regular variation. *Journal of the Royal Statistical Society*, 44:37–42.

Harchol-Balter, M., Crovella, M., and Murta, C. (1998). On choosing a task assignment policy for a distributed server system. In *Proceedings of Performance Tools '98*, pages 231–242. Springer-Verlag.

Hill, B. (1975). A simple general approach to inference about the tail of a distribution. *Annals of Statistics*, 3:1163–1174.

Hogg, T., Huberman, B., and Williams, C. (1996). Phase Transitions and Complexity. *Artificial Intelligence*, 81.

Hoos, H. (1998). PhD Thesis,TU Darmstadt.

Hoos, H. (2000). Personal communications.

Horvitz, E., Ruan, Y., Gomes, C., Kautz, H., Selman, B., and Chickering, M. (2001). A Bayesian Approach to Tackling Hard Computational Problems. In *Proceedings of the Seventeenth Conference On Uncertainty in Artificial Intelligence (UAI-01)*.

Huberman, B., Lukose, R., and Hogg, T. (1993). An economics approach to hard computational problems. *Science*, (265):51–54.

Ilog (2001a). Ilog cplex 7.1. user's manual.

Ilog (2001b). Ilog solver 5.1. user's manual.

Johnson, D. and Trick, M. (1996). Cliques, coloring, and satisfiability: Second dimacs implementation challenge. In *Dimacs Series in Discrete Mathematics and Theoretical Computer Science, Vol. 36*.

Kamath, A., Karmarkar, N., Ramakrishnan, K., and Resende, M. (1990). Computational experience with an interior point algorithm on the satisfiability problem. In *Proceedings of Integer Programming and Combinatorial Optimization*, pages 333–349, Waterloo, Canada. Mathematical Programming Society.

Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., and Selman, B. (2002). Dynamic Restart Policies. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, Edmonton, Canada. AAAI Press.

Kautz, H., Ruan, Y., Achlioptas, D., Gomes, C., and Selman, B. (2001). Balance and Filtering in Structured Satisfiable Problems. In *Proceedings of the Seventeenth Inernational Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, WA.

Kautz, H. and Selman, B. (1996). Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*. AAAI Press.

Kautz, H. and Walser, J. (1999). State-space planning by integer optimization. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, Orlando, FA. AAAI Press.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, (220):671–680.

Kirkpatrick, S. and Selman, B. (1994). Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264:1297–1301.

Kumar, S. R., Russell, A., and Sundaram, R. (1999). Approximating latin square extensions. *Algorithmica*, 24:128–138.

Kwan, A. (1995). Validity of normality assumption in csp research. In *Proceedings of the Pacific Rim International Conference on Artificial Intelligence*, pages 459–465.

Laywine, C. and Mullen, G. (1998). *Discrete Mathematics using Latin Squares*. Wiley-Interscience Series in Discrete mathematics and Optimization.

Leyton-Brown, K., Nudelman, E., and Shoham, Y. (2002). Learning the empirical hardness of optimization problems: the case of combinatorial auctions. In *Principles and Practice of Constraint Programming CP-2002*.

Li, C. and Anbulagan (1997). Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Nagoya, Japan.

Li, C. M. (1999). A constrained-based approach to narrow search trees for satisfiability. *Information processing letters*, 71:75–80.

Luby, M., Sinclair, A., and Zuckerman, D. (1993). Optimal Speedup of Las Vegas Algorithms. *Information Process. Lett.*, pages 173–180.

Lynce, I., Baptista, L., and Marques-Silva, J. (2001). Stochastic systematic search algorithms for satisfiability. In *LICS Workshop on Theory and Applications of Satisfiability Testing (LICS-SAT)*.

Lynce, I. and Marques-Silva, J. (2002a). Building state-of-the-art sat solvers. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*.

Lynce, I. and Marques-Silva, J. (2002b). Complete unrestricted backtracking algorithms for satisfiability. In *Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT'02)*.

Lynce, I. and Marques-Silva, J. (2002c). The effect of nogood recording in mac-cbj sat algorithms. In *ERCIM/CologNet Workshop on Constraint Solving and Constraint Logic Programming, June 2002*.

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

Mandelbrot, B. (1963). The variation of certain speculative prices. *Journal of Business*, 36:394–419.

Mandelbrot, B. (1983). *The fractal geometry of nature*. Freeman: New York.

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

Martin, J., Cox, S., and MacMinn, R. (1988). *The Theory of Finance: Evidence and Applications*. The Dryden Press.

McAloon, K. and Tretkoff, C. (1997). Sports League Scheduling. In *Proceedings of the Third ILOG Conference*, Paris, France.

Meier, A., Gomes, C., and Mellis, E. (2001). An Application of Randomization and Restarts to Proof Planning. In *Proceedings of the Sixth European Conference On Planning (ECP-01)*.

Mitchell, D. and Levesque, H. (1996). Some pitfalls for experimenters with random SAT. *Artificial Intelligence*, 81(1–2):111–125.

Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535.

Motwani, R. and Raghavan, P. (1995). *Randomized algorithms*. Cambridge University Press.

Nemhauser, G. and Trick, M. (1998). Scheduling A Major College Basketball Conference. *Operations Research*, 46(1):1–8.

Prosser, P. (1993). Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299.

Rabin, M. (1980). Probabilistic algorithm for testing primality. *J. Number Theory*, 12:128–138.

Régin, J.-C. (1994). A filtering algorithm for constraints of difference in CSPs. In *Proceedings of The Twelfth National Conference on Artificial Intelligence*, pages 362–367.

Regin, J. C. (1999). Constraint Programming and Sports Scheduling Problems. Informs.

Regin, J. C. (2002). Personal communications.

Reiss, R. and Thomas, M. (2001). *Statistical Analysis of Extreme Values with Applications to Insurance, Finance, Hydrology and Other Fields*. Birkhäuser.

Rish, I. and Frost, D. (1997). Statistical analysis of backtracking on inconsistent csps. In *Proceedings of Third International Conference on Principles and Practices of Constraint Programming*, pages 150–162.

Ruan, Y., Kautz, H., and Horvitz, E. (2002). Restart Policies with Dependence among Runs: a Dynamic Programming Approach. In *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming (CP02)*, Ithaca, USA.

Samorodnitsky, G. and Taqqu, M. (1994). *Stable Non-Gaussian Random Processes: Stochastic Models with Infinite Variance*. Chapman and Hall.

Savelsbergh, M. W. P. (1994). Preprocessing and probing for mixed integer programming problems. *ORSA Journal on Computing*, (6):445–454.

Selman, B. and Kautz, H. (1993). Local Search Strategies for Satisfiability Testing. In *Proceedings of DIMACS Workshop on Maximum Clique, Graph Coloring, and Satisfiability*.

Selman, B. and Kirkpatrick, S. (1996). Finite-Size Scaling of the Computational Cost of Systematic Search. *Artificial Intelligence*, 81(1–2):273–295.

Shaw, P., Stergiou, K., and Walsh, T. (1998). Arc consistency and quasigroup completion. In *Proceedings of ECAI-98, workshop on binary constraints*.

Smith, B. and Grant, S. (1995). Sparse constraint graphs and exceptionally hard problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 646–651. AAAI Pess.

Solovay, R. and Strassen, V. (1977). A fast monte carlo test for primality. *SIAM Journal on Computing*, 6:84–86.

Trick, M. (1996). http://mat.gsia.cmu.edu/color/solvers/trick.c. Source code for the implementation of an exact deterministic algorithm for graph coloring based on DSATUR.

Vandengriend and Culberson (1999). The $G_{nm}$ Phase Transition is Not Hard for the Hamiltonian Cycle Problem. *Journal of Artificial Intelligence Research*.

Veloso, M. (1992). Learning by analogical reasoning in general problem so lving. Ph.D. Thesis, CMU, CS Techn. Report CMU-CS-92-174.

Vossen, T., Ball, M., Lotem, A., and Nau, D. (1999). Integer Programming Models in AI Planning. Technical report, Workshop on Planning as Combinatorial Search, held in conjunction with AIPS-98, Pittsburgh, PA.

Walsh, T. (1999). Search in a Small World. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Stockholm, Sweden.

Watts, D. J. and Strogatz, S. (1998). Collective Dynamics of *Small World* Networks. *Nature*, 393:440–442.

Williams, R., Gomes, C., and Selman, B. (2003a). Backdoors to typical case complexity. In *To appear in Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*.

Williams, R., Gomes, C., and Selman, B. (2003b). On the connections between backdoors, restarts, and heavy-tailedness in combinatorial search. In *Proceedings of Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT-03)*.

Wolfram, S. (2002). *A New Kind of Science*. Stephen Wolfram.

Zhang, H. (2002). A random jump strategy for combinatorial search. In *Proc. of International Symposium on AI and Math*.

Zolotarev, V. (1986). One-dimensional stable distributions. Vol. 65 of "Translations of mathematical monographs", American Mathematical Society. Translation from the original 1983 Russian Ed.