# Randomization and Restart Strategies

by

Huayue Wu

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2006

# Abstract

The running time for solving constraint satisfaction problems (CSP) and propositional satisfiability problems (SAT) using systematic backtracking search has been shown to exhibit great variability. Randomization and restarts is an effective technique for reducing such variability to achieve better expected performance. Several restart strategies have been proposed and studied in previous work and show differing degrees of empirical effectiveness.

The first topic in this thesis is the extension of analytical results on restart strategies through the introduction of physically based assumptions. In particular, we study the performance of two of the restart strategies on a class of heavy tail distribution that has been used to model running time. We show that the geometric strategy provably removes heavy tail of the Pareto form. We also examine several factors that arise during implementation and their effects on existing restart strategies.

The second topic concerns the development of a new hybrid restart strategy in a realistic problem setting. Our work adapts the existing general approach on dynamic strategy but implements more sophisticated machine learning techniques. The resulting hybrid strategy shows superior performance compared to existing static strategies and an improved robustness.

# Acknowledgments

I would like to thank my supervisor, Peter van Beek, for his invaluable guidance and infinite amount of patience. I am deeply grateful for his continual encouragement and the meticulous effort he has spent proofreading this thesis. I would also like to thank Alex Lopez-Ortiz and Prabhakar Ragde for their prompt willingness to act as readers. Their suggestions are insightful and are greatly appreciated. Special thanks go to my parents for without their help and support this thesis would not have been possible. Finally, I would like to thank all the people who had inspired and encouraged me over the years.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Many challenging problems from diverse fields can be easily formulated as optimization problems where the solution(s) sought must satisfy a set of constraints. In this thesis, I am interested in the cases where the application of constraint-based modeling generates a constraint satisfaction problem (CSP) or a satisfiability problem (SAT). One area of active research in computer science has been the study and development of efficient backtracking techniques for solving CSP and SAT problems.

Since CSP and SAT problems are NP-hard, no solution techniques can be expected to have guaranteed good performance. Instead, there is an inherent variation of orders of magnitude in solution time [18]. Some of this variation is attributed to degrees of effectiveness of the backtracking search algorithm. Wrong branches taken early in the search tend to have a catastrophic effect on the search effort [23, 28, 46]. The observation has led to adding randomization to backtracking search algorithms in the hope of avoiding bad decisions during some of the runs [23, 20].

The randomized search can be further combined with a restart scheme where the search is terminated and restarted at regular intervals [2, 23, 35]. The effect is that long runs are aborted early thus saving on the cost of branching mistakes and improving the expected performance. Mathematically, the restart scheme trims the tail in the runtime probability distribution associated with the randomized search [3]. The resulting transformed distribution often has a smaller expected value. This translates into a shorter expected runtime and therefore a faster solver. In practice, the improvement can be very dramatic and some problems are essentially unsolvable without restart [47].

In the past two decades, there have been both theoretical as well as practical advances on the technique of randomization and restart. Several different restart schemes or strategies have been proposed each using a different sequence of restart intervals. These restart strategies have demonstrated varying degrees of effectiveness in practice. Examples include the fixed-cutoff strategy [2, 35], Luby's universal strategy [35], the geometric strategy [45] and the dynamic strategy [26, 30, 40]. Expected performance bounds and optimality conditions are available for the first two strategies.

However, there are still many open questions and issues that serve as motivation for this thesis. For example, theoretical analyses so far have assumed that the runtime distributions of randomized searches can be arbitrary, but in reality these distributions can be more precisely modeled. Moreover, the theories are overly simplified and neglect many practical constraints and factors. Lastly, empirical evaluations of the state-of-the-art restart techniques fail to include realistic situations for which the suggested performance is relatively poor.

## 1.1 Summary of Contributions

The aim of this thesis is to expand on the theoretical framework for randomization and restart as well as to improve on the state-of-the-art restart techniques in terms of both performance and practical applicability.

**Randomization and restart on heavy-tailed distributions.** The fixed-cutoff and geometric restart strategies are analyzed using a heavy-tailed model for the runtime distribution. We derive expressions for the expected performance and variance. We also obtain performance bounds for the geometric strategy and show for the first time that it indeed removes heavy tail.

**Effects of practical constraints on randomization and restart.** We analyze the effects of some of the factors encountered when applying restart techniques in practice. We show that by setting a deadline not only is the optimal infinite strategy no longer optimal but the optimality measure itself needs to change as well. We also derive expressions showing the impact of overhead cost for each restart and a non-trivial minimum effort required in solving real instances.

**Extending and improving dynamic restart techniques.** We introduce a new hybrid static-dynamic strategy. The dynamic strategy portion is based on an existing general approach but extended to a novel problem setup that is more realistic. The approach uses new techniques for feature extraction and predictor inference. The resulting performance is greatly improved. We also address the weakness in the existing evaluation setup and choose to compare performance in measures that are of more practical interest.

## 1.2 Outline of the Thesis

This thesis is organized as follows:

Chapter 2 introduces the background material needed to understand the general topic discussed in this thesis and is intended for readers not familiar with the subject area. Concepts such as CSP and SAT are introduced. We also present the systematic backtracking search algorithms used to solve these problems and show how and why they can be randomized. We then give an overview of the concept of runtime distribution and heavy tail distribution that are central to the randomization and restart techniques. We finish with a quick review of the machine learning techniques used later in the thesis.

Chapter 3 reviews the existing work on the subject of randomization and restart. We begin with a description of the motivation and limitations of the technique. We then present several of the most well-known restart strategies and the relevant theoretical results.

Chapter 4 is a collection of new theoretical results we have derived for the static restart strategies. The first available analytical results on geometric strategies are obtained by assuming a heavy-tailed model for the runtime distribution. We also discuss how the theories change when realistic factors are considered.

Chapter 5 presents our new hybrid restart strategy that combines the geometric and dynamic restart strategies and is shown to be robust. We describe the feature extraction and predictor inference in our approach that give a great boost to the runtime prediction accuracy. The problem setup we use is more realistic but for which results are missing from previous work. We also compare existing strategies on performance measures that are of more practical concern.

Chapter 6 summarizes the thesis and suggests some possible future work.

# Chapter 2

# Background

There have been many developments and ideas in the long history of CSP and SAT problems. This chapter provides a short review of those concepts that form the basis of the work discussed in this thesis. We first introduce CSP and SAT problems and describe how they can be solved with systematic backtrack search. We then recap the basic ideas from probability theory as they relate to algorithm randomization. We give a primer on heavy-tailed distributions which in the following chapter is shown to be central to the idea of randomization and restart. We finish with a brief look at machine learning techniques.

## 2.1 CSP, SAT and Systematic Search

Formally, a CSP is defined by a 3-tuple $(V, D, C)$ where $V$ is the set of variables, $D$ the set of domains for the variables, and $C$ the set of constraints whose form is problem dependent. A solution to a CSP is an $n$-tuple $((v_1, a_1), (v_2, a_2), .., (v_n, a_n))$ where each variable $v_i$ is assigned a value $a_i$ from its domain such that all the constraints are simultaneously satisfied. If there is no solution then the CSP is unsatisfiable.

**Example 2.1.** The N-queens problem has traditionally been used to demonstrate how CSP modeling works. The problem is to place N queens on an N×N chess board such that no two queens attack each other. In one possible model the variables are the rows, and the domain for each variable consists of the column numbers. The constraints are that no two queens attack each other. A solution to the problem is a set of ordered pairs of row and column numbers that specify placement of the queens satisfying the given constraints. In this simple problem, all constraints are binary, that is, involving only two variables (two rows).

For general CSPs, the number of variables in a constraint can be arbitrary and is referred to as the arity of the constraint. The multitude of terminologies on CSP are thoroughly explored in reference materials (see, e.g. [42]).

SAT problems form a special class of CSP problems. They have small variable domains containing only two values, true and false, and the constraints are Boolean clauses given in conjunctive normal form (CNF). In contrast to more general CSP problems, SAT problems have a long history and have been relatively well investigated. Some of the results are generalizable to other CSP domains but many are specific to SAT problems.

### 2.1.1 Solution Techniques

Many techniques have been proposed for solving CSP and SAT problems, including local search and hybrid algorithms, but the dominant technique is backtracking search. The term "backtrack" was first introduced in the 1950's by D.H. Lehmer [5], and the first backtrack search algorithm was credited to Davis and Putnam [10] and later improved by Davis, Longemann and Loveland [9] to become the well-known DPLL algorithm. The family of backtrack algorithms for solving CSPs saw many enhancements and modifications since the 80's. Bitner and Reingold [5] in the 70's applied the technique to solve several small applications and in the process identified several important algorithmic aspects of backtrack search including pre-processing (later known as propagation or filtering), symmetry breaking, and variable ordering. Haralick and Elliott in 1980 [22] gave the first comprehensive analysis of several algorithmic extensions such as look-ahead and backmarking. Since then, many other techniques were introduced to greatly improve the performance of backtrack search. More recently, Chen and van Beek [7] established and proved a performance dominance hierarchy among various backtrack algorithms.

The general structure of the backtrack algorithm is given in Figure 2.1. In each recursion, the algorithm uses heuristics to pick an unbound variable and a value to assign to it. A propagator and look-ahead module prunes the search space as much as possible in light of the new assignment. If inconsistency is detected, the algorithm backtracks and tries an alternative value assignment. When alternative value assignments for the current variable are exhausted the algorithm then backtracks further to an earlier variable. Otherwise, the recursive search continues with a new unbound variable. The search terminates when either arriving at a solution where all variables are bound (i.e. the problem becomes empty), or all branches in the search tree are shown to lead to inconsistency and the problem is proved unsatisfiable. In practice, however, there is often an execution time limit and the problem is indeterminate when neither state is reached before the deadline.

The most elusive components of the search algorithm are the heuristics. Although much effort has been spent on developing more sophisticated heuristics, there does not exist a simple dominance hierarchy. No heuristic performs universally well across all problem domains and performance can differ by orders of magnitude across instances. This observation is not surprising considering CSP and SAT are NP-complete problems. Often, the choice of heuristic decides if a particular instance can be solved in any reasonable amount of time.

```
Function Search($CSP$)
        If (Propagate($CSP$) = failure) return Unsatisfiable
        If ($CSP$ is empty) return Satisfiable
        Choose_Variable($CSP$)
        While (Choose_and_Assign_Value($CSP$) $\neq$ exhausted)
                status := Search($CSP$)
                If (status = Satisfiable) return Satisfiable
        return Unsatisfiable
```

Figure 2.1: Basic backtrack search algorithm.

## 2.2 Randomized Algorithms

A randomized algorithm is an algorithm whose computational process possesses a degree of randomness that is reflected in a performance measure such as the runtime. There are two kinds of randomized algorithms, Las Vegas algorithms in which the solution is correct but the runtime varies, and Monte Carlo algorithms whose runtime is guaranteed but the result is allowed a certain error bound. The motivation behind the Las Vegas type is the hope of achieving good average performance. This is particularly relevant in our context where CSPs can seem to take forever to solve when using a deterministic solver.

A backtrack search algorithm can be turned into a Las Vegas type algorithm, for example, by introducing randomness into the heuristic decisions. The runtime of the resulting algorithm on any given instance can be represented by a random variable, say $T$. Recall from probability theory that a random variable has an associated probability distribution that assigns a probability to an event, which in this case is the length of execution. Figure 2.2 shows a sample runtime distribution of a randomized backtrack search algorithm on a crossword puzzle instance.

The runtime distribution can be uniquely described by its cumulative distribution function $F(t)$, the probability that the algorithm stops on or before time $t$, and is defined as,

$$F(t) = P[T \leq t].$$

One can also speak of the probability mass function in the discrete case representing the probability the algorithm stops at time $t$ and is defined as,

$$f(t) = P[T = t],$$

and the probability density function for the continuous case is defined implicitly as,

$$P[a \leq T \leq b] = \int_a^b f(t)dt.$$

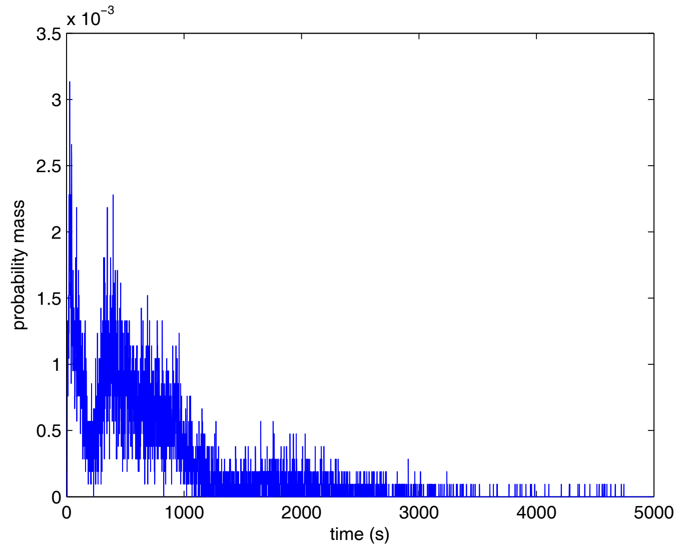

Figure 2.2: Runtime distribution for a randomized solver on a 15×15 crossword puzzle instance. The x-axis is the runtime $T$ given in seconds and the y-axis is $P[X = t]$.

Like other probability distributions, the runtime distribution can be characterized by properties such as the expected value $E[T]$ and the variance $Var[T]$. Another interesting measure is the tail probability function or survival function representing the probability the algorithm would take time longer than $t$ to finish and is defined as,

$$P[T > t] = 1 - F(t).$$

Lastly, the hazard function specifies the instantaneous finishing rate as a function of time and is given by,

$$h(t) = \frac{f(t)}{1 - F(t)}.$$

## 2.3 Heavy Tail

Heavy tail was a concept first introduced in economics by Pareto and later expanded on by Levy and Mandelbrot [36, 37]. The heavy-tailed model has found application in modeling a wide variety of physical and sociological phenomena from power grid failures to Internet traffic patterns, and, in our case, the runtime distribution of a randomized search algorithm. Heavy tail refers to a probability distribution having a slow-decaying or fat tail when compared with the classic exponentially decaying tail. An intuitive interpretation is that rare events are not all that rare. Specifically, a heavy-tailed distribution obeys the following:

$$P[T > t] \sim Ct^{-\alpha}, \quad \text{where } t \to \infty, 0 < \alpha < 2, C > 0.$$

As the algebraic form suggests, heavy tail is also known as *power-law* decay where $\alpha$ is referred to as the *index of stability*. The interesting property of a heavy-tailed distribution is that when $\alpha \leq 1$ the distribution has neither finite mean nor finite variance. When $1 < \alpha < 2$, the distribution has only finite mean. And for $\alpha \geq 2$ the distribution has both finite mean and finite variance. A particularly simple heavy-tailed distribution is the Pareto distribution whose probability mass function and cumulative distribution function are given by,

$$f(t) = \alpha t^{-\alpha-1}, \quad \alpha > 0,$$
$$F(t) = 1 - t^{-\alpha}, \quad \alpha > 0.$$

Heavy tail is best illustrated graphically with a *log-log* plot where the logarithm of the tail $(1 - F(t))$ is plotted against $\log(t)$. For fast decaying distributions such as the exponential distribution, one would observe a sharp drop-off in the tail. But for heavy-tailed distribution, the drop-off is approximately linear and the slope provides an estimate for the index $\alpha$. Figure 2.3 illustrates the scenario.

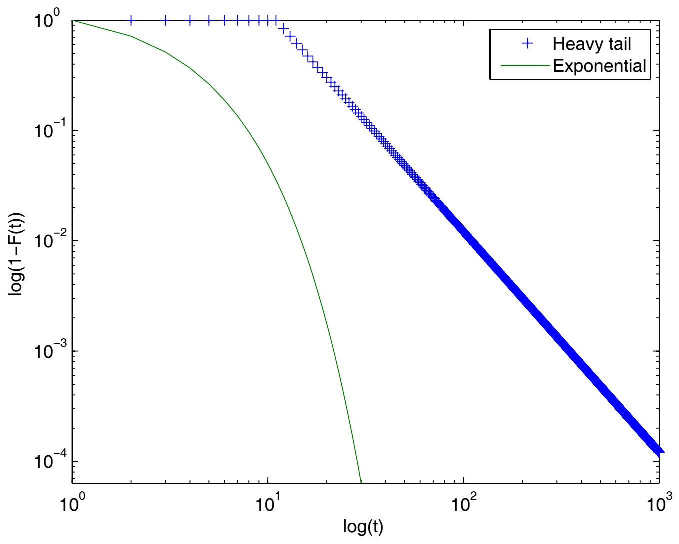

Figure 2.3: Heavy-tailed distribution vs. exponential distribution in $\log - \log$ plot.

In the next chapter we will see that heavy tail can be used to model the runtime distribution of a randomized backtrack search and serves as motivation for the technique of randomization and restart.

## 2.4 Supervised Learning

Later in the thesis, we introduce algorithms that make use of runtime predictors constructed with supervised learning. Supervised learning is a machine learning technique for building a predictor function for a set of data. The general procedure is given in Figure 2.4.

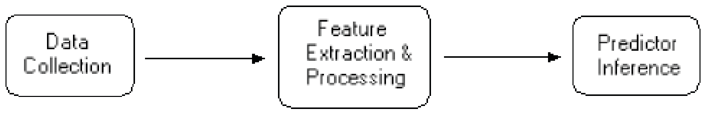

Figure 2.4: General workflow of supervised learning.

Prior to data collection, a problem domain is first determined for which we want to make predictions about a particular objective measure. For example, when solving scheduling problems with randomized backtracking search we may want to predict the runtime length. During data collection, observations and measurements are made on a dataset containing samples that are representative of the given problem domain. These observations include both the objective measure and other information deemed useful in helping make the prediction. For example, the size of a scheduling problem is a useful piece of information since larger problems tend to take longer to solve.

During feature extraction and processing, feature vectors are first formed from the collected set of raw measures. The newly formed features can be either just the raw measures, such as problem size, or a combination of different raw measures. One feature vector is formed for each of the samples in the original dataset. The features are then filtered to keep the feature set small. Such processing is necessary for effective learning because of the curse of dimensionality. That is, the ratio of the number of features to the number of samples must be kept small. Also, highly correlated or redundant features have an impact on predictor robustness and should be removed. There is a large literature (e.g.,[21, 29, 31]) on the topic of feature selection.

Before predictor inference, it is necessary to decide on the form of the predictor function, and an error metric. An example of a predictor function is the decision tree. This is a tree structured graph where the internal nodes are labeled with branching criteria in terms of the features and the leaf nodes are labeled with class names. An error metric used in evaluating the decision tree is the classification error which counts the number of misclassified samples. Once the form of the predictor function and the error metric are fixed, an appropriate learning algorithm can be applied. Available algorithms for learning decision trees include *CART*, *ID3*, and *C4.5* [11].

To assess the effectiveness of the learned predictor, the original dataset is often split into a training set, a validation set, and a test set. The training set is used by the learning algorithm to build the predictor and the test set is used to estimate its typical performance. In iterative inference procedures where a series of predictors of increasing sizes are learned, the validation set is used for picking the predictor with the appropriate level of complexity. Such a predictor can be expected to perform well on future unseen data without overfitting, i.e. tuned too much to the peculiarities of the given training set.

## 2.5 Summary

Many combinatorial problems can be easily and naturally formulated as CSP and SAT problems. These problems can be solved with systematic backtrack search where performance is largely influenced by the heuristics used in deciding variable and value branchings. The deterministic search can be randomized through, for example, the heuristics. The resulting Las Vegas type algorithm has an associated runtime distribution and can be studied with ideas from probability theory. One type of runtime distribution that arises is the heavy-tailed distribution that is characterized by a slow decaying asymptotic tail and it may not have finite mean and variance. Machine learning techniques are useful for predicting runtime length and consist of three major steps including data collection, feature extraction and predictor inference. In the next chapter we will introduce the idea and techniques of randomization and restart making use of the concepts introduced here.

# Chapter 3

# Existing Restart Strategies

This chapter reviews existing work on randomization and restart. We begin by describing the motivations behind the technique and a chronological recount of major developments. We then discuss the conditions under which randomization and restart is useful as well as providing intuitive explanations for why heavy tails occur. After enumerating the various ways of randomizing a backtrack search algorithm we explain in more details the prominent restart strategies. Specifically, we review the fixed-cutoff, Luby's universal, geometric, Frisch's combined and dynamic restart strategies and provide theoretical results when available. We also briefly mention the algorithm portfolio idea that is closely related to randomization and restart.

## 3.1 Motivation and Background

Backtrack search algorithms have been widely observed to exhibit great variability in runtime for seemingly small changes in the variable and value ordering heuristics. The reason is that heuristics are not perfect and do make mistakes. The severity of the consequence depends on the number as well as the timings of the mistakes and is reflected in the observed runtime. A randomized backtrack solver would show a similar variability in performance due to the built-in perturbations.

The technique of randomization and restart was proposed to take advantage of such variability and help improve average performance. The idea is to capitalize on the fact that runtime distributions in practice have non-trivial probability mass for small $t$ that is associated with solving an instance quickly, as shown in Figure 2.2.

Specifically, one would introduce restarts in the backtrack algorithm where during each run the algorithm stops once a certain cutoff threshold is exceeded and restarts the search from the beginning with a different random seed until the problem is solved. The modified algorithm is given in Figure 3.1.

One refers to a particular sequence of cutoff values as a restart strategy and there are many things to consider in an implementation. For example, the cutoff value can be a

```
Function Search($CSP$)
    If (Cutoff_Reached()) return Timed_out
    If (Propagate($CSP$) = failure) return Unsatisfiable
    If ($CSP$ is empty) return Satisfiable
    Choose_Variable($CSP$)
    While (Choose_and_Assign_Value($CSP$) $\neq$ exhausted)
        status := Search($CSP$)
        If (status $\neq$ Unsatisfiable) return status
    return Unsatisfiable


Function Restart_Wrapper($CSP$)
    While (Search($CSP$) = Timed_out)
        Reinitialize()
        Choose_Seed()
```

Figure 3.1: Backtrack search algorithm with restart.

constant or differ from run to run, and the number of restarts can be either finite or infinite. Note that the completeness of the algorithm is no longer guaranteed in general when the cutoff value is fixed. It is possible to have a cutoff value that is so small that the probability of finding a solution is zero even when one exists. Similarly, the cutoff value may not be enough to finish exploring a minimal refutation tree when the problem is unsatisfiable. In either case the algorithm will never terminate. One solution is to have cutoffs that increase over time so that a complete search is performed asymptotically. Moreover, it is also necessary to have an infinite number of restarts to guarantee completeness.

The technique of introducing randomization and restart into the backtrack search algorithm started at least as early as in the work of Harvey [23]. Harvey found that the consequences of early mistakes can be largely avoided by restarting the backtrack search periodically with different variable orderings. The observation led Harvey to propose a randomized algorithm in which the search terminates when the distance backtracked from a dead-end exceeds some fixed cutoff and is restarted with a different ordering until the problem is solved. The new algorithm, an implementation of a fixed-cutoff strategy, was shown to give improved performance over a deterministic backtrack search algorithm on job shop scheduling problems.

At the same time, in the study of Las Vegas algorithms, Alt et al. [2, 3] were one of the first to analyze the effects on the tail when applying infinite restarts to arbitrary runtime distributions. They proved that there exists a probabilistic strategy guaranteeing an exponentially decaying tail and they obtained a slightly weaker result on deterministic strategies. Luby et al. [35] later expanded on this early work and showed that for any runtime distribution there exists a fixed-cutoff infinite strategy that is optimal. They also proposed a universal strategy and proved that it is optimal amongst all possible universal strategies.

Gomes et al. [18, 19, 20] have since done much to popularize and advance the restart technique within backtrack search algorithms. Their large body of work on this topic includes demonstrations of the wide applicability of restarts, drawing connections to existing work on Las Vegas algorithms, as well as contributions to an understanding of when and why restarts help.

To intuitively understand the effect of restarts, it helps to graphically examine the relationship between the original probability distribution and that which is transformed by a restart strategy. Suppose we have an input distribution with cumulative distribution function $F(t)$ and we apply a restart strategy with a sequence of cutoffs $(t_1, t_2, t_3, \dots)$. The resulting cumulative distribution function $F'(t)$ can then be defined recursively as,

$$F'(t) = \begin{cases} F(t) & t \leq t_1 \\ F'(t_i) + (1 - F'(t_i))F(t - t_i) & t_i < t \leq t_{i+1}. \end{cases}$$

In other words, the final probability mass function is comprised of successive copies of segments of the original distribution with lengths $(t_1, t_2, t_3, \dots)$. Each segment is scaled down by how much of the cumulative probability remains at the restart point. An example of such a transformation is illustrated in Figure 3.2.

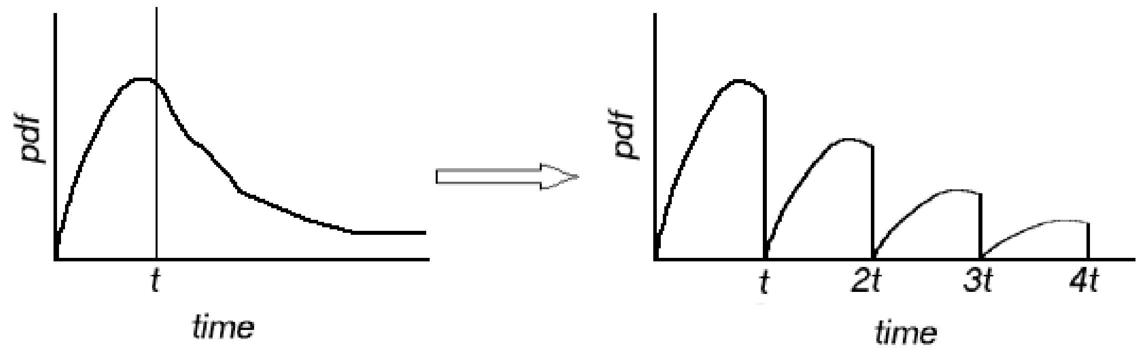

Figure 3.2: Effect of restart strategy on runtime distribution when the sequence of cutoffs is $(t, t, t, \dots)$.

One thing to note is that the randomness introduced in converting the deterministic solver into a Las Vegas algorithm will sometimes weaken the heuristics. When the heuristic is particularly good for the problems at hand, it should not be surprising that the expected performance of even an optimal restart strategy can be inferior to that of the deterministic solver.

## 3.2 When Do Restarts Help?

This section examines the questions of when and why restart strategies are helpful from two angles: For what kinds of runtime distributions are restarts useful and what are the underlying causes for these runtime distributions?

### 3.2.1 Runtime Distributions For Which Restarts Are Useful

Gomes et al. [18] were the first to suggest the use of heavy tail in the modeling of the observed runtime distribution, specifically on those problems that are neither too trivial nor too hard. Figure 3.3 gives a sample *log-log* plot of such a runtime distribution in solving a crossword puzzle instance. Gomes et al. showed that randomization and restarts were particularly effective on such distributions through the elimination of the tail section and capitalizing on the early probability mass. It should be noted, however, that the problems observed to exhibit heavy tail are all satisfiable problems and it has been suggested that unsatisfiable instances do not have heavy tails and so may not benefit from randomization and restarts [4].

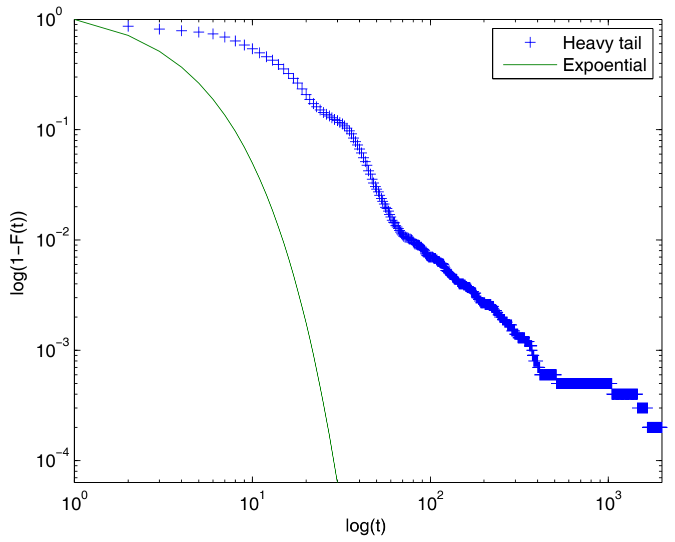

Figure 3.3: Heavy-tailed distribution in solving crossword puzzle.

Hoos [25] observed that restarts will not only be effective for heavy tails, but that its effectiveness depends *solely* on there existing some point where the cumulative runtime distribution is increasing slower than the exponential distribution. It is at this point, where the search is most likely to be stagnating, that a restart would be helpful.

Van Moorsel and Wolter [43] provided a necessary and sufficient condition for restarts to be useful. Their work can be seen as a formalization of Hoos' insight and its extension from one restart to multiple restarts. Let $T$ be a random variable that models the runtime of a randomized backtracking algorithm on an instance. Let $E[T]$ be the expected value of $T$. Under the assumption that successive runs of the randomized algorithm are statistically independent and identically distributed, van Moorsel and Wolter showed that *any* number of restarts using a fixed cutoff of $t$ is better than letting the algorithm run to completion if and only if,

$$E[T] \quad < \quad E[T - t \mid T > t],$$

holds; i.e., if and only if the expected runtime of the algorithm is less than the expected remaining time to completion given that the algorithm has run for $t$ steps. Ó Nualláin et al. [38] obtained a similar but more explicit condition stating that restart should be performed at time $t$ whenever,

$$E[T] \quad < \quad \frac{\sum_{t'>t} t' f(t')}{1 - F(t)} - t.$$

Of course, these conditions are only sensible when the expected value of the original runtime distribution actually exists.

Van Moorsel and Wolter observed that heavy-tailed distributions would satisfy this condition for at least some values of $t$ whereas for an exponential distribution the condition becomes equality implying that restarts neither help nor hurt. In practice, however, there is a cost associated with initializing each restart, so the performance in the latter case is in fact worse. Zhan [47] showed that restarts can indeed be harmful for many problems.

### 3.2.2 Underlying Causes For Heavy Tail

Various theories have been postulated for explaining why restarts are helpful; i.e., why runtime distributions arise for which restarts are beneficial. It is superficially agreed that an explanation for this phenomenon is that ordering heuristics make mistakes [18]. For example, a critical error in an early branching decision could lead the search into an unsatisfiable part of the search tree that is exponentially large and would take an impractical amount of time to refute. However, the theories differ in what it means for an ordering heuristic to make a mistake.

Harvey [23] defines a mistake as follows.

**Definition 3.1.** *A mistake is a node in the search tree that is a nogood but the parent of the node is not a nogood.*

A nogood is a node where the sub-problem is unsatisfiable. When a mistake is made, the search has branched into a subproblem that does not have a solution. The result is that the node has to be refuted and doing this may require a large subtree to be explored, especially if the mistake is made early in the tree. In this definition, value ordering heuristics make mistakes, variable ordering heuristics do not. However, changing the variable ordering can mean either that a mistake is not made, since the value ordering is correct for the newly chosen variable, or that any mistake is less costly to correct. Harvey constructed a probabilistic model to predict when a restart algorithm will perform better than its deterministic counterpart. With simplifying assumptions about the probability of a mistake, it was shown that restarts are beneficial when the mistake probability is small.

As evidence in support of this theory, Hulubei and O'Sullivan [28] considered the distribution of refutation sizes to correct mistakes (the size of the subtrees that are rooted at mistakes). They showed that when using a poor value ordering in experiments on quasigroup completion problems, heavy-tailed behaviour was observed for every one of four different high-quality variable ordering heuristics. However, the heavy-tailed behaviour disappeared when the same experiments were performed but this time with a high-quality value ordering heuristic in place of the random value ordering.

Williams, Gomes and Selman [46] (hereafter just Williams) define a mistake as follows.

**Definition 3.2.** *A mistake is a selection of a variable that is not in a minimal backdoor, when such a variable is available to be chosen.*

A backdoor is a set of variables for which there exists value assignments such that the simplified problem (such as after constraint propagation) can be solved in polynomial time. Backdoors capture the intuition that good variable and value ordering heuristics simplify the problem as quickly as possible. When a mistake is made, the search has branched into a subproblem that has not been as effectively simplified as it would have been had it chosen a backdoor variable. The result is that the subproblem is more costly to search, especially if the mistake is made early in the tree. In this definition, variable ordering heuristics make mistakes, value ordering heuristics do not. Williams constructs a probabilistic model to predict when heavy-tailed behaviour would occur as well as when there will exist a restart strategy with polynomial expected runtime. With simplifying assumptions about the probability of a mistake, it was shown that both of these occur when the probability of a mistake and the size of the minimal backdoor are sufficiently small. The theory can also explain when restarts would be beneficial for unsatisfiable problems, through the notion of a strong backdoor. However, the theory does not entirely account for the fact that a random value ordering together with a restart strategy can remove heavy-tailed behaviour. In this case the variable ordering remains fixed and so the probability of a mistake also remains unchanged.

Finally, there is some work that contributes to an understanding of why runtime distributions arise where restarts are helpful while remaining agnostic about the exact definition of a mistake. Consider the probability distribution of refutation sizes to correct mistakes. It has been shown both empirically on random problems and through theoretical, probabilistic models that heavy tails arise in the case where this distribution decays exponentially as the size of the refutation grows [6, 15]. In other words, there is an exponentially decreasing probability of making a costly (exponentially-sized) mistake.

## 3.3 Randomization Techniques

There are several different ways of introducing randomization into a deterministic backtrack search algorithm. Harvey [23] was the first to propose randomizing the variable ordering. Gomes et al. [20, 19] suggest specific strategies for randomizing a variable ordering heuristic by either breaking ties or picking from top scorers randomly. They show that with restarts the randomized algorithm led to orders of magnitude improvement on a wide variety of problems from both SAT and CSP versions of scheduling, planning, and quasigroup completion problems. Other alternatives are to choose a variable with probability proportional to the heuristic score or to pick randomly from a suite of different heuristics.

One pitfall to be aware of is that the randomization method must give enough different choices near the top of the search tree for maximum effectiveness. For example, simple random tie-breaking sometimes is not ideal because the heuristic scores are sufficiently distinct near top of the tree. Harvey [23] also proposed randomizing the *value* ordering so that each possible ordering is equally likely. The randomization techniques for variable ordering heuristic would apply equally well in this case.

Zhang [48], however, pointed out that randomizing the heuristic would also weaken its discriminating power, an undesirable side effect. He instead proposed an alternative way of randomization called random jump method. The idea is simply to skip large subtrees randomly during systematic search. Specifically, if by estimate a subtree cannot be exhaustively searched during a restart interval then it is considered too large and is skipped, and the notion of large depends on each cutoff value. The completeness of the search is guaranteed as the cutoff value increases and fewer subtrees are skipped.

A shortcoming of the random jump approach is that it is much more difficult to analyze. The runtime distribution depends critically on the quality of the estimate for the subtree sizes which at best is a challenging problem in itself. Moreover, there is no longer a separation between the runtime distribution of the randomized algorithm and that after applying the restart strategy. Therefore one is unable to easily study the general effect of a restart strategy independent from the problem instance.

## 3.4 Restart Strategies

Implementing a restart strategy involves running a randomized algorithm up to some cutoff time, stopping it, and then repeating the process until the problem is solved or is given up after a deadline has passed. The intuition is to take advantage of the left mass of the run-time distribution and trim off long runs. Formally, a restart strategy $S$ is defined by a sequence of cutoffs:

$$S = (t_1, t_2, t_3, \dots).$$

In practice, one also needs to decide on the primitive operation or time unit with which to measure cutoffs. Harvey [23] used the distance the algorithm has backtracked from a dead-end. Gomes et al. [19] counted the total number of backtracks. Kautz et al. [30, 40] measured the number of nodes or choice points visited. In addition, the natural measurement of time in seconds can also be used. The latter is of more practical interest since runtime performance is often the true objective measure and it may be only weakly correlated with the other measurements. This weak correlation is due to different amounts of processing done at nodes located at different depths of the search tree.

There are many different restart strategies depending on how the cutoffs change, and they can be classified as either static or dynamic. Static strategies are parameterized where a sequence of cutoffs is described by a few parameters and is fixed before the search begins. The optimal parameters depend on the combination of problem instance and the algorithm. Dynamic strategies on the other hand generate the sequence of cutoffs at runtime.

The following subsections present an overview of several classes of existing restart strategies and highlights the available theoretical results. From now on, we denote the runtime random variable, the probability density function, and the cumulative distribution function for the original runtime distribution and that after applying a restart strategy by $T$ and $T_S$, $f(t)$ and $f_s(t)$, $F(t)$ and $F_s(t)$ respectively. One assumption for the static strategy is that the number of restarts is unbounded and the strategy is infinite. Another assumption is that each restart is independent.

## 3.4.1 Fixed-cutoff Strategy

Fixed-cutoff strategies are simple static strategies of the form $S = (t_c, t_c, t_c, \dots)$ where $t_c$ is a constant. It is easy to show that a fixed-cutoff strategy removes heavy tail as shown in Theorem 3.1.

**Theorem 3.1.** *For arbitrary distribution f(t), the strategy* $(t_c, t_c, t_c, \dots)$ *guarantees an exponential tail.*

*Proof.* The result follows from the inequality,

$$\begin{aligned} P[T_S > t] &\leq P[T_S > t_c n], \quad \text{where } n = \lfloor \frac{t}{t_c} \rfloor \\ &= (1 - F(t_c))^n \\ &\leq (1 - F(t_c))^{\frac{t}{t_c} - 1}. \end{aligned}$$

Similarly we can show the tail is also bounded below by an exponential distribution. □

The removal of the heavy tail guarantees the existence of the mean for $T_S$ as given in Theorem 3.2. Van Moorsel and Wolter have derived similar results independently in [43]. Luby et al. [35] obtained a different expression that can be shown to be equivalent: $E[T_S] = \frac{1}{F(t_c)}(t_c - \sum_{t' < t_c} F(t'))$.

**Theorem 3.2.** *The mean of the runtime for fixed-cutoff strategy with cutoff $t_c$ is,*

$$E[T_S] \quad = \quad \frac{t_c(1-F(t_c)) + E_{t_c}}{F(t_c)},$$

*where $E_{t_c} = \int_0^{t_c} tf(t)dt$.*

*Proof.* The expression $E_{t_c}$ is the partial mean of the original distribution on the interval $[0, t_c]$. It is also the mean of the first segment in the runtime distribution after applying the restart strategy as shown in Figure 3.2. Each subsequent segment is of equal length but with successively smaller area. The partial mean for the $(m+1)^{st}$ segment is given by,

$$\begin{aligned}
E_{t_c}^m &= \int_0^{t_c} (mt_c + t)f(t)(1-F(t_c))^m, \quad \text{where } m = 0, 1, \ldots \\
&= mt_c(1-F(t_c))^m \left(\int_0^{t_c} f(t)dt\right) + (1-F(t_c))^m \left(\int_0^{t_c} tf(t)dt\right) \\
&= mt_c(1-F(t_c))^m F(t_c) + (1-F(t_c))^m E_{t_c}.
\end{aligned}$$

The derivation of the mean then becomes straightforward:

$$\begin{aligned}
E[T_S] &= \int_0^{\infty} tf(t)dt \\
&= \sum_{m=0}^{\infty} E_{t_c}^m \\
&= F(t_c)t_c \left(\sum_{m=0}^{\infty} m(1-F(t_c))^m\right) + E_{t_c}\left(\sum_{m=0}^{\infty}(1-F(t_c))^m\right) \\
&= F(t_c)t_c \left(\frac{1-F(t_c)}{F(t_c)^2}\right) + \left(\frac{E_{t_c}}{F(t_c)}\right) \\
&= \frac{t_c(1-F(t_c)) + E_{t_c}}{F(t_c)}.
\end{aligned}$$

For a sanity check, observe that $t_c \to 0 \Rightarrow F(t_c) \to 0 \Rightarrow E[T_S] \to \infty$ by l'Hopital's rule. □

With the expression for the mean we can apply simple calculus to find an implicit expression for the optimal fixed cutoff $t_c^*$. Van Moorsel and Wolter [43] also obtain a similar result independently.

**Theorem 3.3.** *The cutoff $t_c^* > 0$ is optimal when,*

$$E[T_S] \quad = \quad \frac{1-F(t_c^*)}{f(t_c^*)}.$$

*Proof.* We take the derivative of the mean with respect to $t_c$ and set it to 0 and obtain,

$$\begin{aligned}
0 &= \frac{d}{dt_c}E[T_S] \quad \Leftrightarrow \\
0 &= \frac{F(t_c)((1-F(t_c)) - t_c f(t_c) + t_c f(t_c)) - (t_c(1-F(t_c)) + E_{t_c})f(t_c)}{F(t_c)^2} \quad \Leftrightarrow \\
F(t_c)(1-F(t_c)) &= (t_c(1-F(t_c)) + E_{t_c})f(t_c) \quad \Leftrightarrow \\
\frac{1-F(t_c)}{f(t_c)} &= \frac{t_c(1-F(t_c)) + E_{t_c}}{F(t_c)} \quad \Leftrightarrow \\
\frac{1-F(t_c)}{f(t_c)} &= E[T_S].
\end{aligned}$$

$\square$

The simple intuitive interpretation is that the optimal strategy is obtained when the expected runtime after applying the restart strategy equals the inverse hazard rate of the original distribution at the cutoff point. Luby et al. [35] proved that $t_c^*$ always exists and the resulting strategy is optimal in the absolute sense that no other static restart strategy of any kind can do better.

Van Moorsel and Wolter [44] observed that for some runtime distributions, such as the *lognormal* distribution, a wide range of cutoffs performed well. They further observed that it was safer for the cutoff to be too large rather than too small.

Figure 3.4 shows an example of how the performance of the fixed-cutoff strategy on a heavy-tailed distribution changes as a function of the cutoff values.

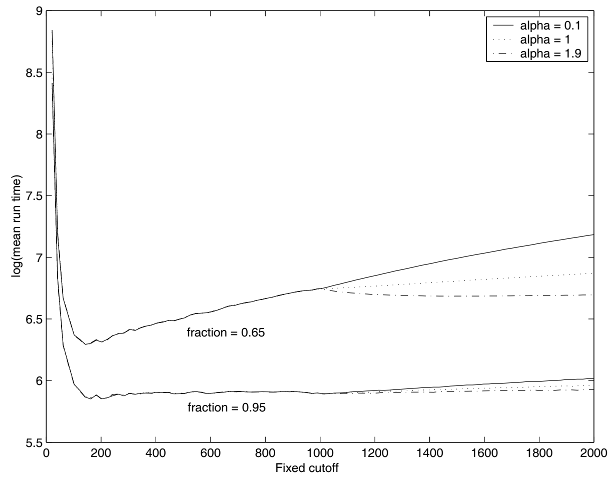

Figure 3.4: Simulation result showing the performance of fixed-cutoff strategy. Input runtime distributions are constructed by attaching a Pareto tail to an empirical runtime distribution from solving a crossword puzzle instance (using $|dom|/deg$ heuristic where ties are broken randomly). The total probability mass given to the tail section is determined by $(1-$*fraction*$)$. Three Pareto tails of different $\alpha$ values are used for each *fraction* setting.

### 3.4.2 Luby's Universal Strategy

The fixed-cutoff strategy is not robust in the sense that one must find a good cutoff value either through trial-and-error or by having knowledge about the runtime distribution. Luby et al. [35] asked the question of how well one can do when absolutely no information is known or can be accrued about the original runtime distribution. They were able to devise a universal strategy based on the fixed-cutoff strategy but with a guaranteed performance bound.

Luby's universal strategy $S = (t_1, t_2, t_3, \dots)$ is defined by the recurrence equations [35]:

$$t_i = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ t_{i-2^{k-1}+1} & \text{if } 2^{k-1} \leq i < 2^k - 1. \end{cases} \tag{3.1}$$

and when expanded the strategy looks like,

$$S = (1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \dots).$$

In form the strategy resembles a mixed set of fixed-cutoff strategies with exponentially increasing cutoffs, as if they are running in "parallel". This highlights the underlying idea that the universal strategy is searching for the optimal cutoff but at the same time keeping the running time taken up by each of the embedded fixed-cutoff strategies roughly equal. By the time the optimal cutoff (or something slightly larger) is found and the corresponding optimal fixed-cutoff strategy has executed enough times to give a high stopping probability, the total amount of time spent is bounded and surprisingly small, as given in Theorem 3.4.

**Theorem 3.4** (Luby et al. [35]). *For arbitrary distribution f(t),*

$$E[T_S] \leq 192 l^* (\log_2(l^*) + 5)$$

*where $l^*$ is the expected runtime of the optimal fixed-cutoff strategy on distribution f(t).*

Since the essence is in keeping a balanced effort between the "parallel" set of fixed-cutoff strategies, the factor 2 in Equation 3.1 can in fact be replaced with any positive integer and the asymptotic performance guarantee would still hold, albeit with a different logarithmic base. This factor or Luby factor can be considered a parameter for the family of generalized Luby universal strategies. Luby et al. also gave a tail bound for their new strategy shown in Theorem 3.5. The tail is not quite exponential, as in the case of fixed-cutoff strategy, but it decays significantly faster than heavy tail.

**Theorem 3.5** (Luby et al. [35]). *The probability that Luby's universal strategy runs for more than t steps is at most,*

$$exp\{-t/64 l^* \log_2(t)\}.$$

Luby et al. [35] completed the development of the universal strategy by proving that the bound in Theorem 3.4 is the best that can be achieved by any universal strategy on arbitrary unknown distributions up to a constant factor. The proof was based on a constructed family of pathological distributions. The question of how well Luby's universal strategy performs on specific classes of runtime distributions remains open. Some of our empirical studies suggest that the performance depends greatly on the choice of parameters and can differ by orders of magnitude. In some cases, even a small change in the parameters produced large differences in performance. A common observation is that the sequence of cutoffs in Luby's universal strategy grows too slowly and often does not greatly improve performance [19, 30, 40].

The basic form of Luby's universal strategy can be extended by introducing a scaling factor $1/s$ to account for different time scales. Figure 3.5 shows an example of how the performance of Luby's universal strategy changes as a function of the Luby factor when applied to a heavy-tailed input distribution.

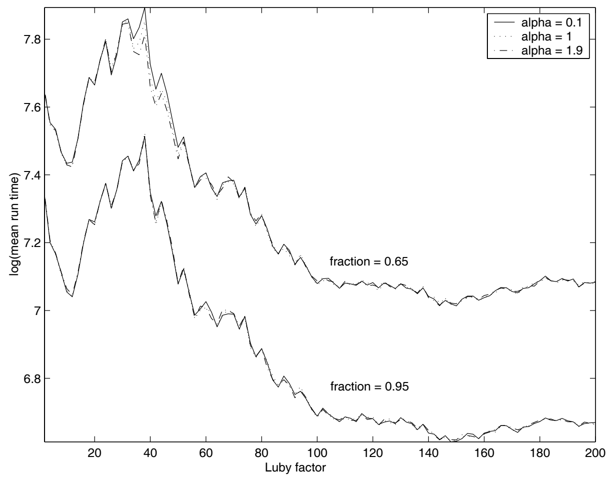

Figure 3.5: Simulation result showing the performance of Luby's universal strategy, where the scale factor is set to 1. Input runtime distributions are constructed by attaching a Pareto tail to an empirical runtime distribution from solving a crossword puzzle instance (using $|dom|/deg$ heuristic where ties are broken randomly). The total probability mass given to the tail section is determined by $(1-fraction)$. Three Pareto tails of different $\alpha$ values are used for each *fraction* setting.

### 3.4.3 Geometric Strategy

In response to the sometimes dismal performance of Luby's universal strategy on practical problems, Walsh [45] proposed a new strategy that adopted the idea of exponentially increasing cutoffs but abandoned the tedious repetitions. The new strategy seems to work robustly in many situations and give good performance gains. The geometric strategy has the general form,

$$S = (r, r^2, r^3, \dots),$$

where geometric factor $r$ controls how fast the cutoff values grow. It is always assumed that $r > 1$ since a shrinking sequence of cutoffs does not guarantee algorithm completeness or remove heavy tail.

Although the geometric strategy can be argued to have been derived from Luby's universal strategy, it has discarded the central idea needed to guarantee a performance bound, namely that of finding the optimal cutoff and executing it repeatedly.

Although it has been observed that an optimally tuned geometric strategy sometimes performs slightly inferior to an optimally tuned Luby's universal strategy, the geometric strategy shows remarkable robustness and tends to work well over a large range of geometric factors. In practice, a geometric factor in the range $1 < r < 2$ often works well on both SAT and CSP instances [45, 47]. This may be a result of the peculiarities of the runtime distributions of many real world problems in that they require no more than a handful of restarts to finish with high probability.

As for Luby's universal strategy, the geometric strategy can also be extended to include a scaling factor $1/s$. Figure 3.6 shows an example of how the performance of the geometric strategy changes as a function of just the geometric factor when applied to a heavy-tailed input distribution.

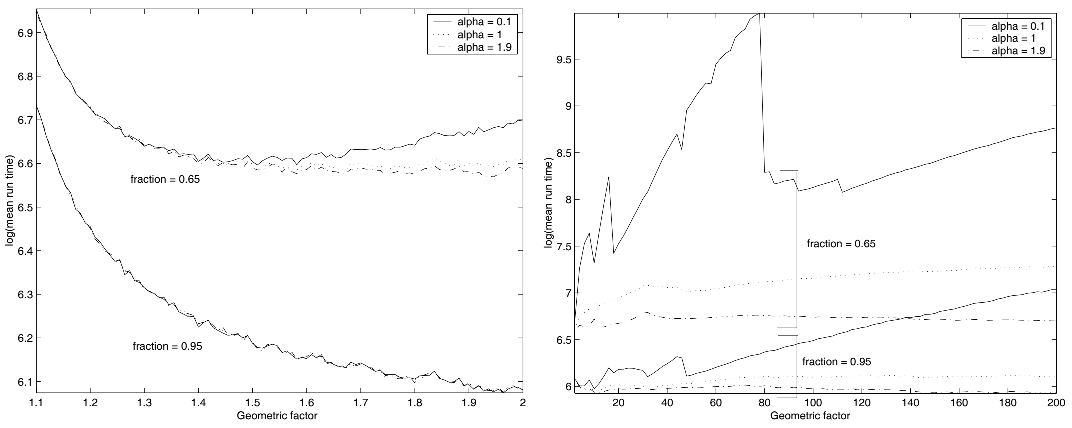

Figure 3.6: Simulation result showing the performance of geometric strategy, where the scale factor is set to 1. Input runtime distributions are constructed by attaching a Pareto tail to an empirical runtime distribution from solving a crossword puzzle instance (using $|dom|/deg$ heuristic where ties are broken randomly). The total probability mass given to the tail section is determined by (1−*fraction*). Three Pareto tails of different $\alpha$ values are used for each *fraction* setting.

More results on the geometric strategy can be found in the next chapter.

### 3.4.4 Frisch's Combined Strategy

A novel restart strategy proposed by Frisch and Zhan [13] is one that combines Luby's universal and the geometric strategies. The basic Frisch's combined strategy has two parameters, one for the geometric factor and one for the repetition count in Luby's universal strategy. For example, a Frisch's combined strategy with geometric factor of 3 and a repetition factor of 2 would look like,

$$S = (1, 1, 3, 1, 1, 3, 9, 1, 1, 3, 1, 1, 3, 9, 27, 1, \dots).$$

Frisch's combined strategy can also be extended to include a scaling factor as the third parameter.

Frisch's combined strategy is interesting because it is a superclass that encompasses all three parameterized strategies discussed above. By setting both the geometric and the repetition factor to 1, we have the fixed-cutoff strategy. Luby's universal strategy is obtained by setting both the geometric and the repetition factor to 2. The geometric strategy is constructed with a repetition factor of 1.

Although an elegant construct, the added parameter dimension makes analysis especially difficult. So far, no theoretical results are available and it has not seen much practical use either. One thing we have observed in preliminary experiments is that the performance of Frisch's combined strategy is often dominated or is the same as that of the geometric strategy when the geometric factor is larger than the repetition factor. When the relative sizes of these two parameters are reversed the performance tends to become much worse, and Frisch's combined strategy starts to resemble the fixed-cutoff strategy. In this case, the performance is governed by the scale parameter which is roughly equivalent in effect to the fixed cutoff value.

### 3.4.5 Dynamic Strategy

Static strategies are simple to implement and generally help improve performance on difficult problems. The theoretical results and worst case bounds provide a peace of mind to some degree. However, even with the same asymptotic guarantees, being able to solve a problem in under an hour and in over a day have different feasibility implications to practitioners. The real question then is: how much better can we do on real problems? Some of the assumptions made in the study of static strategies severely limit their effectiveness and deserve a re-examination.

Kautz et al. [30, 40] noted that two of the assumptions may not hold in practice. First, the assumption that the successive runs of the randomized algorithm are statistically independent and identically distributed becomes invalid in the case where each run is drawn from one of two distributions. Second, the assumption that no prior knowledge is available on the runtime distribution or on each individual run is false as there are an abundance of indirect observations.

Generally, real world problems are not as pathological as in some of the theoretical constructions but instead show remarkable and consistent structures. Knowledge about what distinguishes these problems as coherent subclasses out of all possible problems is available and can be made useful in improving the effectiveness of restarts in a context specific fashion. A systematic approach that has been proven in practice in extracting such knowledge is based on correlation ideas. For problems that are closely related, we would expect there to be some similarities in runs that finish early and also in runs that seem to go on forever, as well as a marked distinction between the two types of runs.

The extraction of features for quantifying such correlation can be done with machine learning techniques. There are basically two sources of information available. Statically, one can look at aspects of the SAT or CSP encodings such as the number of variables and constraints, the sizes of variable domains and constraints, as well as some graph theoretical measures of the constraint relationships. Dynamically, one can observe how the search progresses over time and how some of the data structures change and evolve. Generally, the static information is much less helpful and only serves as a supplement to the dynamic observations. For example, static features can sometimes give an indication of the inherent hardness of an instance and help in biasing the likelihood of long and short runs.

Having established a runtime classifier, it is then straightforward to build a restart strategy based on it. The algorithm would be restarted whenever a run is predicted to take a long time to finish. The resulting dynamic restart strategy would improve the empirical performance over a static strategy whenever the runtime prediction is sufficiently accurate.

Kautz et al. [26] investigated the problem of building runtime classifiers for quasigroup completion problems using Bayesian models. The features they gathered included summary statistics of generic as well as domain specific attributes recorded over an observation horizon of 1000 search tree nodes. For example, one attribute was the number of Boolean variables assigned the value *true* and the summary statistics were the initial, minimum, maximum and average values. Machine learning technique is applied to the collected features and a decision tree is constructed.

The reported prediction accuracy was 60% for single-instance setup and 72% for multi-instance setup. The single instance setup is one in which the same instance is used in all the restarts whereas in the multi-instance setup a different problem is drawn randomly each time a restart takes places. The subtle implication of the multi-instance setup is that the result may be severely skewed. It is possible that the easy instances are solved over and over again but the hard one are simply skipped. This is not a realistic scenario since we are often more interested in the number of unique instances that can be solved.

In a series of later papers, Kautz et al. developed various hybrid static-dynamic restart strategies based on the runtime classifier. The multi-instance setup was used in all of their work. In one paper [30], instances were picked independently from an ensemble of sub-classes during each run. The algorithm was restarted when the classifier predicted the run to be long. Otherwise, the search continued up to an optimal cutoff calculated offline and then restarted. In another paper [40] the independence assumption was dropped and instances were picked from the same sub-distribution each time. A Markov chain model was used to update the belief at each restart about from which sub-distribution the instances came from. The corresponding optimal cutoffs are then applied as appropriate. In all these experiments, Kautz et al. observed significant performance improvements in terms of expected runtime when compared to Luby's universal strategy and even the optimal fixed-cutoff strategy on the entire problem set.

Another related study of note is that of Leyton-Brown et al. [32] for predicting hardness of combinatorial auction problems. Their technique used purely static features including many of interest to general CSP problems such as clustering coefficient, minimum path length and many other graph based measures. The prediction model was based on linear regression. Although allowing a finer degree of discrimination when compared to decision trees, the continuous models are often much worse in terms of robustness particularly on a mixed collection of runtime distributions.

## 3.5 Other Related Work

An idea closely related to restart strategy is that of algorithm portfolio. Instead of running a single randomized algorithm repeatedly, an algorithm portfolio consists of several distinct algorithms. The two types of algorithm portfolios are multiprocessor portfolio and single processor portfolio. In the multiprocessor setup, different numbers of each algorithm are run on a corresponding number of processors. In the single processor setup, not only can one have several copies of a given algorithm but one is also able to adjust the fraction of CPU time assigned to each copy. These two portfolio parameters have very different effect and scale the component distributions vertically and horizontally respectively. The runtime distribution for the entire portfolio is then the summation of these scaled component distributions. Thus a single processor portfolio may not have a simple corresponding multiprocessor portfolio.

**Example 3.1.** Consider a setup with no deadline where the portfolio runs until one of the algorithms find a solution. A simple single processor portfolio is one consisting of two algorithms $A_1$ and $A_2$ that are given 1/3 and 2/3 of CPU time respectively and run concurrently with context switching. The runtime distribution would generally be different qualitatively from that of a multiprocessor portfolio where $A_1$ runs on one CPU and two copies of $A_2$ run on one CPU each.

Hogg and Williams [24] looked at the case of applying a single-processor portfolio to hard computational problems and observed a counter-intuitive result. When the runtime distribution of one algorithm is not strictly dominated by that of another, then it is often beneficial to keep such an algorithm in the portfolio even though its mean is much worse than the others. This result was further confirmed by Gomes and Selman [17, 18]. The intuitive explanation is that, as with restart strategies, portfolios prefer risk-seeking algorithms that contribute to the early probability mass. This trend becomes increasingly evident when the number of algorithms allowed in the portfolio increases. Lastly, we note that portfolios, although they do not provably remove heavy tails, may be more robust than restart strategies due to the added variability in algorithm choices.

## 3.6 Summary

This chapter has presented the general technique of randomization and restart. This technique is effective at diminishing the consequences of early heuristic mistakes. Specifically, restarts help on runtime distributions that decay more slowly than the exponentials. An example is the heavy-tailed distribution. Randomization can be introduced through the heuristics or by skipping parts of the search tree.

The first restart strategy studied was the fixed-cutoff strategy that guarantees an exponential tail and can be made optimal for arbitrary known runtime distributions. Luby's universal strategy was based on the fixed-cutoff strategy with guaranteed asymptotic performance on arbitrary unknown distributions. The geometric strategy tackles the problem of slow growth in Luby's universal strategy and often has more robust empirical performance. All three static strategies can be seen as special cases of Frisch's combined strategy.

Dynamic restart strategies are able to outperform static ones by using information gathered during runtime. Machine learning techniques can be used to classify the length of each run on which the restart decisions are based, and the resulting predictor is used to guide the selection of cutoffs to use.

An idea related to randomization and restart is that of algorithm portfolio. It shares with restart strategies the common underlying principle for improving performance, namely in the preference for risk seeking algorithm; i.e., in taking advantage of early mass in the runtime distribution.

In the next two chapters, we will present new results we have obtained on the static and dynamic strategies, respectively.

# Chapter 4

# New Results on Static Restart Strategies

Previously, restart strategies were studied with the assumption of arbitrary runtime distributions. In this chapter, we present new results that expand the theoretical framework by focusing on a practically relevant class of distributions, the heavy-tailed distributions. We analyze the fixed-cutoff strategy and provide bounds on the expected performance. For the geometric strategy we show that its performance in general is not bounded with respect to the optimal fixed-cutoff strategy. We also prove for the first time that it removes heavy tail. Later in the chapter, we examine in detail several idealized assumptions made in previous studies about deadline, restart overhead cost and minimum solution effort. We assess the effect and impact of removing these assumptions and the resulting observations are of interest to practitioners.

## 4.1 Fixed-cutoff Strategy

It has been mentioned previously in Section 3.2.1 that the runtime distributions encountered in solving real world instances can be modeled reasonably well with heavy tails [19]. It is therefore of interest to develop the theoretical results further using this special class of runtime distributions. Specifically, we shall examine the pure Pareto distribution introduced in Section 2.3. Although this simple model does not capture the intricate details in real world runtime distributions prior to the tail section, it nevertheless allows us to make qualitative observations and to obtain asymptotic results.

**Theorem 4.1.** *For the Pareto distribution with density function $f(t) = \alpha t^{-\alpha-1}$, $\alpha > 0$, $\alpha \neq 1$, $t > 1$, the expected runtime after applying a fixed-cutoff strategy with cutoff $t$ is given by,*

$$E[T_S] = \frac{t(\alpha t^{\alpha-1} - 1)}{(t^{\alpha} - 1)(\alpha - 1)}.$$

*Proof.* From Theorem 3.2,

$$\begin{aligned}
E[T_S] &= \frac{1}{F(t)}\left(t(1-F(t)) + E_t\right) \\
&= \frac{1}{F(t)}\left(t(1-F(t)) + \int_1^t t' f(t') dt'\right) \\
&= \frac{1}{1-\frac{1}{t^\alpha}}\left(\frac{1}{t^{\alpha-1}} + \int_1^t t'\alpha t'^{(-\alpha-1)} dt'\right) \\
&= \frac{t^\alpha}{t^\alpha - 1}\left(\frac{1}{t^{\alpha-1}} + \frac{\alpha}{\alpha-1}\left(1 - \frac{1}{t^{\alpha-1}}\right)\right) \\
&= \frac{t(\alpha t^{\alpha-1} - 1)}{(t^\alpha - 1)(\alpha - 1)}.
\end{aligned}$$

Moreover, for a discrete Pareto distribution it can be shown that $E[T_S] < E_{discrete}[T_S] < E[T_S] + 1$. □

As a sanity check, observe that as $t \to \infty$ the strategy becomes degenerate and the runtime distribution approaches that of the Pareto; $E[T_S]$ becomes divergent and unbounded for $\alpha \leq 1$. This agrees with the fact that the Pareto distribution has no finite mean for such $\alpha$ values. When $\alpha > 1$, $E[T_S] \to \frac{\alpha}{\alpha-1}$ as $t \to \infty$ and this is just the mean for the Pareto distribution. Figure 4.1 shows how the function $E[T_S]$ behaves and also how it is similar to the expected runtime plot generated from empirical data in Figure 3.4.

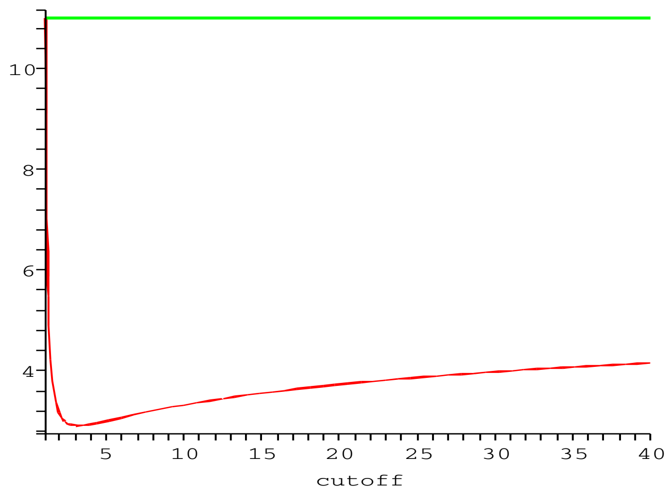

Figure 4.1: The function $E[T_S]$ for increasing cutoffs where $\alpha = 1.1$. The horizontal line shows the asymptote $\frac{\alpha}{\alpha-1}$.

**Theorem 4.2.** *For $0 < \alpha < 2$, $\alpha \neq 1$, the cutoff value that minimizes the function $E[T_S]$ is $t^* \in [2, 4]$. Moreover, when $\alpha > 2$, $|E[T_{S(t^*)}] - E[T_{S(3)}]| < 0.268$ where $T_{S(t)}$ denotes the* *random variable that models the runtime distribution of a fixed-cutoff strategy with cutoff $t$.*

*Proof.* Differentiating $E[T_S]$ we get,

$$\frac{dE[T_S]}{dt} = \frac{-\alpha^2 t^{(\alpha-1)} - (1-\alpha)t^{\alpha} + 1}{(\alpha-1)(t^{\alpha}-1)^2}$$

We want to show that $\frac{dE[T_S]}{dt} < 0$ for $t < 2$ and $\frac{dE[T_S]}{dt} > 0$ for $t > 4$. In other words, $\frac{dE[T_S]}{dt} = 0$ for some $t^* \in [2,4]$.

First consider the case when $1 < t < 2$ and $0 < \alpha < 1$. The denominator is negative. The numerator is an inverted quadratic function in $\alpha$ for any value of $t$ so its minimum value occurs on the interval boundary. Observe that,

$$\lim_{\alpha\to 0^+} -\alpha^2 t^{(\alpha-1)} - (1-\alpha)t^{\alpha} + 1 = 0^+$$

and,

$$\lim_{\alpha\to 1^-} -\alpha^2 t^{(\alpha-1)} - (1-\alpha)t^{\alpha} + 1 = 0^+$$

Thus, $\frac{dE[T_S]}{dt} < 0$ in this case.

Next, consider the case when $1 < t < 2$ and $1 < \alpha < 2$. The denominator is positive. We want to find the maximum value for the numerator so we first differentiate it with respect to $\alpha$ and get,

$$\frac{\partial(numerator)}{\partial\alpha} = -2\alpha t^{\alpha-1} - \alpha^2 t^{\alpha-1}\ln(t) + t^{\alpha} - (1-\alpha)t^{\alpha}\ln(t)$$

Setting to 0 and solving for $\alpha$ we get,

$$\alpha(t) = \frac{-2 + \ln(t)t + \sqrt{4 + \ln(t)^2t^2 - 4\ln(t)^2 t}}{2\ln(t)} > 0$$

With the help of *Maple*, we find that $\alpha(t) < 1$ over the interval $1 < t < 2$. Thus, the maximum value for the numerator occurs on the interval boundary of $\alpha$. We observe that,

$$\lim_{\alpha\to 1^+} -\alpha^2 t^{(\alpha-1)} - (1-\alpha)t^{\alpha} + 1 = 0^-$$

and,

$$\lim_{\alpha\to 2^-} -\alpha^2 t^{(\alpha-1)} - (1-\alpha)t^{\alpha} + 1 < 0$$

Thus, $\frac{dE[T_S]}{dt} < 0$ in this case as well.

Similarly, we find that $\frac{dE[T_S]}{dt} > 0$ for $t > 4$ and the first result follows.

When $\alpha \geq 2$, the difference $|E[T_{S(t^*)}] - E[T_{S(3)}]|$ is given by,

$$\int_2^{t^*} \left| \frac{dE[T_S]}{dt} \right| \quad < \quad \int_2^{\infty} \left| \frac{dE[T_S]}{dt} \right|$$

which is maximum when $\alpha = 2$ and is less than 0.268 shown using *Maple.* □

Theorem 4.2 states that over the range of $\alpha$'s of interest, the optimal cutoff is very much constant and is close to the value 3. When $\alpha$ is much larger and heavy tail is less of a problem, a cutoff value in the narrow range of $[2, 4]$ will still give performance that is bounded closely to that of the optimal cutoff. This is in spite of the fact that the optimal cutoff approaches $\infty$ as $\alpha$ tends to $\infty$ .

However, the result does not mean that the fixed-cutoff strategy is a robust universal strategy in implementation by always choosing a cutoff value of 3. It is important to remember that the runtime distributions of real problems may be very different from the simple Pareto distribution. They would most notably be stretched or compressed horizontally and have an early probability mass portion that is more complex. Thus, the time unit associated with the magical number 3 would vary greatly from instance to instance. The implication is that if machine learning techniques can be used, for example, to predict the stretch and compression factor needed in transforming a given runtime distribution into a reasonable fit with the Pareto, then there may be a way of applying the fixed-cutoff strategy more robustly in practice. In this case, the arbitrary choice of $\alpha$ can be taken advantage to account to some degree for the general shape of the early probability mass portion. Similarly, we can also consider using a mixture of several Pareto distributions as the template in order to obtain a better fit. In such case, an analogous result to Theorem 4.2 can be derived.

## 4.2 Geometric Strategy

The geometric strategy is qualitatively very different from Luby's universal strategy, as metioned in Section 3.4.3, and a new set of theoretical results are needed to characterize this fast growing strategy. The first notable property we establish in Theorem 4.3 is that the geometric strategy does not have any performance guarantee and is therefore not a universal strategy.

**Theorem 4.3.** *The expected run-time of the geometric strategy can be arbitrarily worse than that of the optimal strategy.*

*Proof.* For any given geometric strategy of the form $(1, r, r^2, \cdots)$, $r > 1$, define a probability distribution as follows,

$$f(t) \quad = \quad \begin{cases} \frac{1}{l} & t = 1 \\ 1 - \frac{1}{l} & t = \infty \\ 0 & otherwise \end{cases}$$

where $l = r/(r-1)$. Note that the optimal fixed-cutoff strategy $(1, 1, 1, \cdots)$ has expected run-time $l$. The expected run-time $E[T_S]$ of the geometric strategy is given by,

$$\begin{aligned}
E[T_S] &= \frac{1}{l} + (1-\frac{1}{l})\frac{(1+1)}{l} + (1-\frac{1}{l})^2\frac{(1+1+r)}{l} + \cdots \\
&= \left(\sum_{i=0}^{\infty}(1-\frac{1}{l})^i\right)\left(\frac{1}{l} + \frac{1}{l}(1-\frac{1}{l}) + \frac{r}{l}(1-\frac{1}{l})^2 + \cdots\right) \\
&= l\left(\frac{1}{l}\right)\left(1 + (1-\frac{1}{l}) + r(1-\frac{1}{l})^2 + \cdots\right) \\
&= 1 + (1-\frac{1}{l})\sum_{i=0}^{\infty}(r-\frac{r}{l})^i \\
&= 1 + (1-\frac{1}{l})\sum_{i=0}^{\infty}1^i.
\end{aligned}$$

Thus the expected run-time $E[T_S]$ is unbounded with respect to that of the optimal fixed-cutoff strategy. □

Notation-wise, the condition $t = \infty$ can also be formulated as $t = r^k$ and a similar result on $E[T_S]$ can be derived by letting $k \to \infty$.

Note that for real CSP and SAT problems there always exists a $t$ for which $P(T \leq t) = 1$; i.e. the runtime distributions are finite. In such case, the result in Theorem 4.3 no longer holds and the performance of the geometric strategy is always bounded. However, such a bound cannot be easily parameterized and depending on the original runtime distribution it can be exponentially large compared to that of the optimal fixed-cutoff strategy.

In a theoretical sense, this initial result in Theorem 4.3 is a setback to the general applicability of the geometric strategy and raises question about its value. As mentioned in Section 3.2.1, the runtime distributions likely encountered in practice are much less pathological and can often be modeled reasonably well with heavy-tailed distributions. It is therefore of theoretical and practical interest to study the behaviour of the geometric strategy in this more specific context. Theorem 4.4 shows, for the first time, that the geometric strategy does indeed remove heavy tail from Pareto distributions and guarantees an expected performance gain. The expressions are given in terms of the strategy parameters which would help in understanding how the performance of the strategy changes with respect to parameter tuning.

**Theorem 4.4.** *For the Pareto distribution with cumulative probability distribution* $F(t) = 1 - Ct^{-\alpha},\ t \geq 1,\ \alpha > 0,\ C > 0$, *the geometric strategy* $S = 1/s(r, r^2, r^3, \ldots)$ *has the following tail,*

$$P[T_S > t] \quad \leq \quad M(st)^{-\frac{\alpha}{2}\log_r st + R} \tag{4.1}$$

*where* $R = \frac{\alpha}{2} + \log_r(s^\alpha C)$, $\quad M = \begin{cases} (s^\alpha C)^{-1} & \textit{for } s^\alpha C < 1 \\ 1 & \textit{for } s^\alpha C \geq 1. \end{cases}$

*Proof.* Without loss of generality, we assume $t \geq \frac{r}{s}$. This helps to simplify the derivation although the general result holds regardless.

The idea is to first bound the tail probability function from above by a piecewise linear function which is further bounded by a smooth function that decays faster than heavy tail.

$$\begin{aligned}
P[T_S > t] &\leq P[T_S > \frac{r^{i_t}}{s}], \quad \textit{where } i_t = \arg\max_i(\frac{r^i}{s} \leq t) = \lfloor \log_r st \rfloor \\
&= \prod_{l=1}^{i_t} \left(1 - F\left(\frac{r^l}{s}\right)\right) \\
&= \prod_{l=1}^{i_t} s^\alpha C r^{-\alpha l} \\
&= (s^\alpha C)^{\lfloor \log_r st \rfloor} \prod_{l=1}^{\lfloor \log_r st \rfloor} r^{-\alpha l} \\
&= (s^\alpha C)^{\lfloor \log_r st \rfloor} r^{-\alpha \sum_{l=1}^{\lfloor \log_r st \rfloor} l}
\end{aligned}$$

Since $\sum_{l=1}^{\lfloor \log_r st \rfloor} l = \frac{1}{2}(1 + \lfloor \log_r st \rfloor)\lfloor \log_r st \rfloor \geq \frac{1}{2}(1 + (\log_r st - 1))(\log_r st - 1) = \frac{1}{2}(\log_r^2 st - \log_r st)$, together with the fact that $r > 1$ we have,

$$\begin{aligned}
P[T_S > t] &\leq (s^\alpha C)^{\lfloor \log_r st \rfloor} r^{-\frac{\alpha}{2}(\log_r^2 st - \log_r st)} \\
&= (s^\alpha C)^{\lfloor \log_r st \rfloor} (st)^{-\frac{\alpha}{2}(\log_r st - 1)}
\end{aligned}$$

In the case where $s^\alpha C < 1$ we have,

$$\begin{aligned}
P[T_S > t] &\leq (s^\alpha C)^{\log_r st - 1} (st)^{-\frac{\alpha}{2}(\log_r st - 1)} \\
&= (st)^{\log_r(s^\alpha C)} (s^\alpha C)^{-1} (st)^{-\frac{\alpha}{2}(\log_r st - 1)} \\
&= (s^\alpha C)^{-1} (st)^{-\frac{\alpha}{2}\log_r st + \left(\frac{\alpha}{2} + \log_r(s^\alpha C)\right)} \\
&= (s^\alpha C)^{-1} (st)^{-\frac{\alpha}{2}\log_r st + R}
\end{aligned}$$

In the case where $s^\alpha C \geq 1$ we have,

$$\begin{aligned}
P[T_S > t] &\leq (s^\alpha C)^{\log_r st} (st)^{-\frac{\alpha}{2}(\log_r st - 1)} \\
&= (st)^{\log_r(s^\alpha C)} (st)^{-\frac{\alpha}{2}(\log_r st - 1)} \\
&= (st)^{-\frac{\alpha}{2}\log_r st + \left(\frac{\alpha}{2} + \log_r(s^\alpha C)\right)} \\
&= (st)^{-\frac{\alpha}{2}\log_r st + R}
\end{aligned}$$

The basic form of the tail probability is $e^{-\log^2 t}$. Although slower than $e^{-t}$, it no longer decays polynomially and thus is not heavy tailed. □

Moreover, it is quick to check that both the mean and variance are finite for the cumulative probability function $P[T \leq t] = 1 - e^{-\log^2 t}$.

The result in Theorem 4.4 can be extended beyond a pure Pareto distribution to include those that have tails following a gradual transition to heavy tail. In such case, we can assume without loss of generality that the tail begins at a switch-over point $t'$. The mean and variance can then be approximated by combining contributions from both the initial and tail sections. Both moments are guaranteed to be finite under the geometric restart strategy.

An interesting question to ask is whether the results can also be generalized to strategies similar in form to the geometric strategy? In particular, what happens when the exponent is a polynomial function? Theorem 4.5 states that this general form of the geometric strategy would also eliminate heavy tail, as long as the polynomial has bounded degree. The mean and variance would also be finite. Of course, the larger the degree the slower the tail decay and the larger the mean and variance.

**Theorem 4.5.** *For the Pareto distribution with cumulative probability distribution* $F(t) = 1 - Ct^{-\alpha},\ t \geq 1,\ \alpha > 0,\ C > 0$*, the geometric strategy* $S = 1/s(r^{h(1)}, r^{h(2)}, r^{h(3)}, \dots)$ *where* $h(\cdot)$ *is a polynomial of degree* $m$ *has tail of the basic form* $e^{-(\log t)^{1+\frac{1}{m}}}$.

The logical follow-up question is how good is the geometric strategy at improving performance on heavy-tailed distributions, for example, when compared to the base-line optimal fixed-cutoff strategy which always exists (Section 3.4.1)? Is there a performance bound similar to that for Luby's universal strategy as given in Section 3.4.2? The question remains open in general but some simple observations can be made. In the case when the value of $\alpha$ is bounded away from 0 there is an upper bound on the mean for any geometric strategy with fixed parameters as shown in Theorem 4.4. In other words, when applying a fixed geometric strategy to the class of distributions where $\alpha \geq \epsilon > 0$ for some fixed $\epsilon$, there is an absolute bound on the performance. However, when $\alpha$ is allowed to approach 0 then the mean for both the geometric strategy and optimal fixed-cutoff strategy would tend to $\infty$, but the relationship between the two is unknown.

We also observe that the optimal geometric strategy on any given problem instance has trivially bounded performance by letting the geometric factor $r$ approach 1 and adjust the scale factor to give the optimal fixed-cutoff. We therefore have constructed a strategy that simulates an optimal fixed-cutoff strategy. Similarly, the worst geometric strategy would trivially have no performance bound by letting the geometric factor $r$ approach $\infty$, effectively removing restarts. These observations apply to other parameterized strategies as well.

## 4.3 Constraints on Parameterized Restart Strategies

In the last chapter, several parameterized families of static restart strategies were studied under idealized conditions. The assumptions and simplifications helped to make the analysis straightforward so that a qualitative idea could be quickly grasped. However, real world situations are not as simple. The factors that have an impact on any practical implementation of restart strategies need to be examined more closely.

First, the restart strategies in Chapter 3 were assumed to continue forever until the instance is solved. In practice, an algorithm would never be allowed the luxury of an indefinite amount of time. The value of the solution often becomes negligible or zero if it cannot be reached in a reasonable amount of time. Second, the cost of performing each restart such as doing memory clean-up and re-initialization can no longer be neglected when the cutoff value is small and the number of restarts numerous. Third, there is always a minimum amount of time required to have any chance of solving a real problem and this $t_{min}$ can have an impact when the cutoff value selected happens to be smaller. The following sections will examine each of these factors in more detail.

### 4.3.1 Deadline

When solving real world problems such as scheduling problems, there is always a realistic deadline $D$ at which point the problem must be abandoned if a solution has not been reached. Assuming the performance measure is still the expected runtime, the previous theoretical results must be re-worked with this consideration in mind.

The fixed-cutoff strategy with deadline needs to be redefined and would have the general form $S_t^n = (t, t, \ldots, t, t_0)$. The new form is comprised of $n$ equal restart values $t$ and a left over bit $t_0 < t$. Here, $n = \lfloor D/t \rfloor$ and $t_0 = D - nt$. The expected running time of the strategy $S_t^n$ for a discrete runtime distribution is given by the recurrence,

$$E[T_{S_t^n}] = \begin{cases} 0 & \text{for } n = 0 \\ \sum_{t'=1}^{t} t' f(t') + (1 - F(t))(t + E[T_{S_t^{n-1}}])) & \text{for } n \geq 1. \end{cases}$$

The solution to the recurrence is given in Theorem 4.6.

**Theorem 4.6.** *When given a deadline $D$, the expected runtime of the fixed-cutoff strategy $S_t^n = (t, t, \ldots, t, t_0)$ where $n = \lfloor D/t \rfloor$ and $t_0 = D - nt$ is given by,*

$$E[T_{S_t^n}] = E[T_S](1 - (1 - F(t))^n) + (1 - F(t))^n \left( t_0 - \sum_{t'=1}^{t_0 - 1} F(t') \right),$$

*where $E[T_S]$ is the expected runtime of the infinite fixed-cutoff strategy $S_t = (t, t, t, \ldots)$.*

*Moreover, the probability a solution is found within the deadline $D$ is given by,*

$$L[T_{S_t^n}] = (1 - (1 - F(t))^n) + (1 - F(t))^n F(t_0).$$

*Proof.*

$$E[T_{S_t^n}] \quad = \quad \sum_{t'=1}^{t} t' f(t') + (1 - F(t))(t + E[T_{S_t^{n-1}}])$$

Since,

$$\begin{aligned}
\sum_{t'=1}^{t} t' f(t') &= t \sum_{t'=1}^{t} f(t') - \sum_{t'=1}^{t} (t - t') f(t') \\
&= tF(t) - \sum_{t'=1}^{t-1} \sum_{j=1}^{t-t'} f(t') \\
&= tF(t) - \sum_{t'=1}^{t-1} \sum_{j=1}^{t-t'} f(j) \\
&= tF(t) - \sum_{t'=1}^{t-1} \sum_{j=1}^{t'} f(j) \\
&= tF(t) - \sum_{t'=1}^{t-1} F(t')
\end{aligned}$$

Substituting and we get,

$$\begin{aligned}
E[T_{S_t^n}] &= tF(t) - \sum_{t'=1}^{t-1} F(t') + (1 - F(t))(t + E[T_{S_t^{n-1}}]) \\
&= t - \sum_{t'=1}^{t-1} F(t') + (1 - F(t))E[T_{S_t^{n-1}}]
\end{aligned}$$

Furthermore, applying the identity $E[T_S] = \frac{1}{F(t_c)}(t_c - \sum_{t'<t_c} F(t'))$ (Luby et al. [35]) we have,

$$E[T_{S_t^n}] \quad = \quad E[T_S]F(t) + (1 - F(t))E[T_{S_t^{n-1}}]$$

Now, we unroll the recurrence equation using the fact that $E[T_{S_t^i}] = E[T_S]F(t) + (1 - F(t))E[T_{S_t^{i-1}}]$ except at the very end where $E[T_{S_t^1}] = E[T_S]F(t) + (1 - F(t)) \left(t_0 - \sum_{t'=1}^{t_0-1} F(t')\right)$

and we get,

$$\begin{aligned}
E[T_{S_t^n}] &= E[T_S]F(t)\left(\sum_{i=0}^{n-1}(1-F(t))^i\right) + (1-F(t))^n\left(t_0 - \sum_{t'=1}^{t_0-1}F(t')\right) \\
&= E[T_S]F(t)\frac{1-(1-F(t))^n}{F(t)} + (1-F(t))^n\left(t_0 - \sum_{t'=1}^{t_0-1}F(t')\right) \\
&= E[T_S](1-(1-F(t))^n) + (1-F(t))^n\left(t_0 - \sum_{t'=1}^{t_0-1}F(t')\right).
\end{aligned}$$

The derivation for the solution probability $L[T_{S_t^n}]$ is similar.

$$L[T_{S_t^n}] = F(t) + (1-F(t))L[T_{S_t^{n-1}}]$$

Again, we unroll the recurrence equation and get,

$$\begin{aligned}
L[T_{S_t^n}] &= F(t)\left(\sum_{i=0}^{n-1}(1-F(t))^i\right) + (1-F(t))^n F(t_0) \\
&= 1-(1-F(t))^n + (1-F(t))^n F(t_0).
\end{aligned}$$

□

The expression for $E[T_{S_t^n}]$ in Theorem 4.6 is a weighted sum where $(1-(1-F(t))^n)$ gives the probability that the search with restart strategy $S_t^n$ returns an answer within the deadline $D$. Van Moorsel and Wolter have independently obtained a similar result in [43] for the simpler case where an integral number of restarts fit within the deadline.

To further explore the relationship between the finite and infinite fixed-cutoff strategies, we can ask the question of whether the optimal cutoff in one is also optimal for the other. In general, the answer is no. If $t^*$ is the optimal cutoff for the infinite strategy and $t^{**}$ that of the finite strategy, then the two optimal cutoffs can be very different as shown in Example 4.1. Examples can also be constructed where $t^{**} > t^*$, but the difference between the two may be tightly bounded in this case. Asymptotically, as $n \to \infty$ or $F(t) \to 1$, we have $t^{**} \to t^*$.

**Example 4.1.** Consider the run-time distribution given by,

$$f(t) = \begin{cases} \frac{1}{m} & t = 1 \\ 1-\frac{1}{m} & t = m \\ 0 & otherwise \end{cases}$$

and note that $t^* = m$.

Consider the case where $m = 100$ and $D = 200$. Although it appears to make more sense if $t^{**} = m$ as well, but actually $t^{**} = 1$, since $E[T_{S_1^{200}}] = 86.6$ and $E[T_{S_{100}^2}] = 99.01$. Interestingly, the solution probability $L[T_{S_1^{200}}]$ is 0.866, whereas $L[T_{S_{100}^2}] = 1$. This illustrates the fact that a strategy minimizing the expected runtime does not in general maximize the probability that the problem is solved, and vice-versa.

It is important to point out that finite fixed-cutoff strategy $S_t^n$ in the new form shown above is no longer optimal in general. There is no guarantee that there exists a strategy $S_t^n$ with the best expected runtime amongst all potential finite static restart strategies. This in part is due to the inability in fitting an integral number of cutoffs in the allotted time. The effect of the left-over piece is best demonstrated with examples. Given a runtime distribution $f(t)$ with $f(2) = 0.4, f(3) = 0.1, f(\infty) = 0.5$ and a deadline $D = 5$. The optimal strategy is found to be $(2, 3)$, an increasing sequence. For a second runtime distribution with $f(1) = 0.05, f(2) = 0.15, f(4) = 0.8$ plus a deadline of $D = 11$. The optimal strategy is $(4, 4, 2, 1)$, a decreasing sequence. In neither example does there exist an optimal strategy of the form $S_t^n$.

The left-over piece only has a localized effect and we may therefore consider the class of augmented fixed-cutoff strategies comprised of equal restarts except at the very end. More precisely, the new form of strategy would be a monotonically decreasing sequence of restart values except for the last cutoff. We conjecture that this generalization would help restore the optimality condition; that is, there always exists a finite strategy of the new form with the best expected runtime. The intuition is that increasing pairs not at the very end of the strategy can always be replaced with a decreasing subsequence so that the mean either improves or remains unchanged. For example, segments such as $1, 1, 2$ can always be replaced by something equivalent or better. The reasoning is that if $1, 1$ is "better" than $2$ then the strategy $1, 1, 1, 1$ can be no worse than $1, 1, 2$. Similarly, if $1, 1$ is "worse" than $2$ then $2, 2$ would be a better alternative. The two new segments are both monotonically decreasing.

A significant result by van Moorsel and Wolter [44] is the observation that the cutoffs in an optimal finite strategy must coincide with equi-hazard intervals. That is, the performance of a restart sequence $(t_1, t_2, \ldots, t_n)$ is a local extremum if and only if,

$$\frac{f(t_1)}{1 - F(t_1)} = \frac{f(t_2)}{1 - F(t_2)} = \cdots = \frac{f(t_n)}{1 - F(t_n)}.$$

A special case that satisfies this condition is when $t_1 = t_2 = \cdots = t_n = D/n$, i.e., when $t_i's$ are equi-distant intervals, which is just the traditional fixed-cutoff strategy. Unfortunately, these strategies do not always give global maxima as already demonstrated. Finding the optimal strategy by enumeration of all possible sequences of equi-hazard intervals would require complete knowledge of the runtime distribution and even then it is computationally prohibitive. However, for the special class of *lognormal* distributions van Moorsel and

Wolter showed by empirical observation that the traditional fixed-cutoff strategies often give global optima and only take polynomial time to find. Such observation may be extended to other distributions of practical interest but at present remains an open question.

For Luby's universal and the geometric strategies, the existing theoretical results given in Section 3.4.2 and 3.4.3 cannot be easily adapted when a deadline is present. The analysis relied entirely on the asymptotic behaviour of the tail which is no longer a sensible notion when a deadline is present. Thus the asymptotic results as well as performance bounds previously derived are not valid anymore. In particular, Luby's universal strategy may not have enough time to find the optimal fixed cutoff by exponential search. Similarly, the embedded optimal fixed-cutoff strategy is unlikely to execute enough times to guarantee sufficient stopping probability.

Lastly, it should be stressed again that when given a deadline, optimality in solution probability does not correspond to optimality in expected runtime. Since the ultimate aim is in solving problems, this new objective measure is really the one that should be used to compare performance in practice. This issue has been consistently overlooked and the mean runtime has become the de facto standard in evaluating effectiveness of restart strategies in existing work. The difference and similarity between the mean runtime and solution probability measure for the fixed-restart strategy with different cutoff values is illustrated in Figure 4.2. The two curves are qualitatively similar and are mirror images of each other. Thus maximizing one tends to minimize the other. However, the two extrema points do not coincide unless the deadline $D$ is sufficiently large. In Example 4.1 this happens around $D > D_0 \sim 10m$. Thus the discrepancy between the two measures is especially severe for small deadline values and should be duly noted.

## 4.3.2 Restart Overhead Cost

Unlike the implicit assumption in Chapter 3, performing restarts is not free in practice and requires restoring the data structures back to their initial states. Depending on the implementation, this may involve resetting pointers, reinitializing variables, and possibly some memory copy operation. The impact of this added cost on the expected performance of a fixed-cutoff strategy is captured in Theorem 4.7.

**Theorem 4.7.** *When an overhead cost $c$ is associated with each restart then the expected run-time of the infinite fixed-cutoff restart strategy $S$ with cutoff $t$ is,*

$$E[T_{S_c}] \quad = \quad E[T_S] + c\left(\frac{1-F(t)}{F(t)}\right),$$

*where $E[T_S]$ is the expected runtime of the infinite fixed-cutoff strategy without overhead cost.*

*Proof.* In the following derivation, $T_{S'_c}$ denotes the same strategy as $T_{S_c}$ except with the

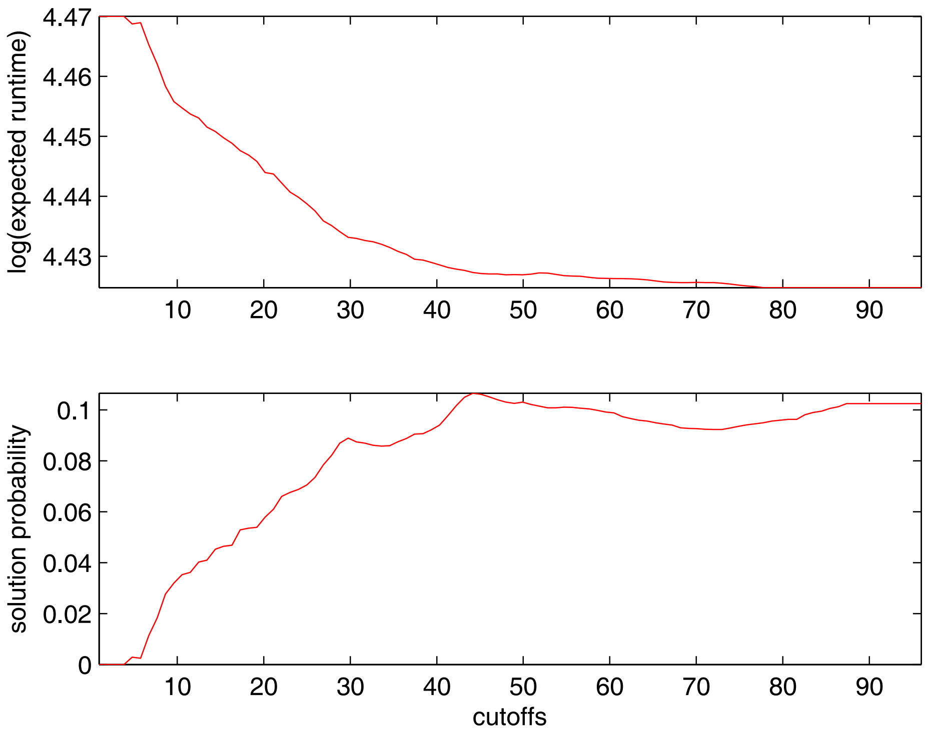

Figure 4.2: Comparison of performance measured in terms of mean runtime and solution probability for different fixed cutoffs. The runtime distribution comes from a crossword puzzle instance and the deadline is 100.

first cutoff omitted.

$$\begin{aligned}
E[T_{S_c}] &= \left(\sum_{t'=1}^{t} t' f(t')\right) + (1 - F(t))(t + c + E[T_{S'_c}]) \\
&= \left(tF(t) - \sum_{t'=1}^{t-1} F(t')\right) + (1 - F(t))t + (1 - F(t))c + (1 - F(t))E[T_{S'_c}] \\
&= t - \sum_{t'=1}^{t-1} F(t') + (1 - F(t))c + (1 - F(t))E[T_{S'_c}].
\end{aligned}$$

We are working with infinite strategies so $T_{S_c} = T_{S'_c}$ and $E[T_{S_c}] = E[T_{S'_c}]$. Thus,

$$\begin{aligned}
F(t)E[T_{S_c}] &= t - \sum_{t'=1}^{t-1} F(t') + (1 - F(t))c \\
E[T_{S_c}] &= \frac{1}{F(t)}(t - \sum_{t'=1}^{t-1} F(t')) + \frac{1 - F(t)}{F(t)}c \\
E[T_{S_c}] &= E[T_S] + \frac{1 - F(t)}{F(t)}c.
\end{aligned}$$

The implication is that as $t \to \infty$ the second term goes to 0 and the effect of the overhead cost diminishes. But for small $t$ values, the cost has a dramatic impact on the expect runtime. The new optimal fixed cutoff will also be greater than the original one. $\square$

For better comparison with the previous result in Theorem 3.2, the expression in Theorem 4.7 can also be written in the form,

$$\frac{(t+c)(1-F(t))+E_t}{F(t)}.$$

The impact of the overhead can be more clearly seen to be on par with that of the cutoff $t$ both of which have equal weighted contribution to the expected runtime. Van Moorsel and Wolter have independently obtained a similar result in [43]. Figure 4.3 shows that as the cutoff $t$ increases the effect of the overhead cost becomes negligible.

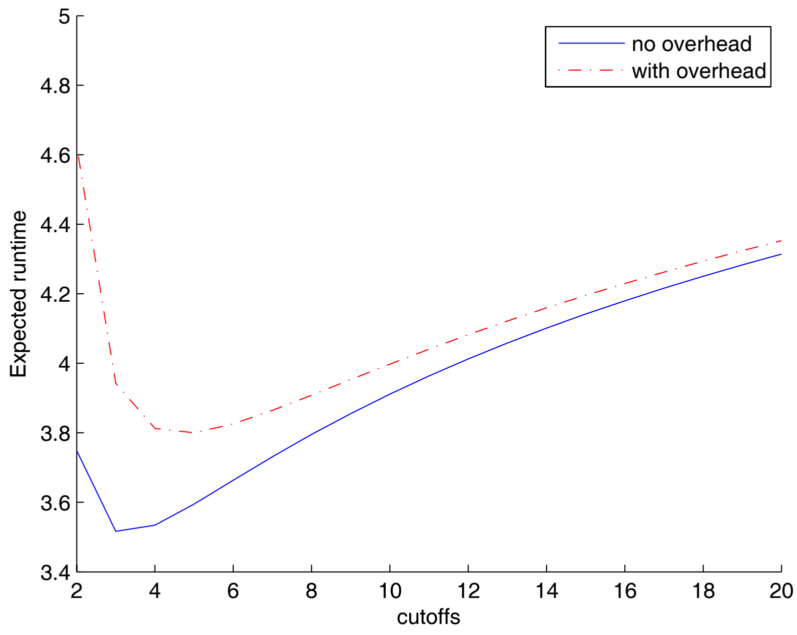

Figure 4.3: Effect of overhead cost $c$ on expected performance of the fixed-cutoff strategy on heavy-tailed distribution.

**Theorem 4.8.** *When given both a deadline $D$ and an overhead cost $c$, the expected run-time of the fixed-cutoff restart strategy $S_t^n = (t, t, \ldots, t, t_0)$ where $n = \lfloor D/t \rfloor$ and $t_0 = D - nt$ is given by,*

$$E[T_{S_t^n}] \quad = \quad \left(E[T_S] + c\frac{1-F(t)}{F(t)}\right)(1-(1-F(t))^n) + (1-F(t))^n \left(t_0 - \sum_{t'=1}^{t_0-1} F(t')\right).$$

*where $E[T_S]$ is the expected runtime of the fixed-cutoff strategy without overhead cost.*

*Proof.*

$$\begin{aligned}
E[T_{S_t^n}] &= \sum_{t'=1}^{t} t' f(t') + (1-F(t))(t+c+E[T_{S_t^{n-1}}]) \\
&= tF(t) - \sum_{t'=1}^{t-1} F(t') + (1-F(t))(t+c+E[T_{S_t^{n-1}}]) \\
&= t - \sum_{t'=1}^{t-1} F(t') + (1-F(t))c + (1-F(t))E[T_{S_t^{n-1}}] \\
&= \left(t - \sum_{t'=1}^{t-1} F(t') + (1-F(t))c\right)\left(1 + (1-F(t)) + \cdots + (1-F(t))^{n-1}\right) \\
&\quad +(1-F(t))^n \left(t_0 - \sum_{t'=1}^{t_0-1} F(t')\right) \\
&= \left(t - \sum_{t'=1}^{t-1} F(t') + (1-F(t))c\right)\frac{1-(1-F(t))^n}{F(t)} + (1-F(t))^n \left(t_0 - \sum_{t'=1}^{t_0-1} F(t')\right) \\
&= \left(E[T_S] + c\frac{1-F(t)}{F(t)}\right)(1-(1-F(t))^n) + (1-F(t))^n \left(t_0 - \sum_{t'=1}^{t_0-1} F(t')\right).
\end{aligned}$$

□

For Luby's universal and the geometric strategies, the essence of the analysis lies in the asymptotic behaviour. Neither the asymptotic tail characterization nor the associated performance bounds would change qualitatively for any finite overhead cost $c$. Thus we omit the lengthy derivation that shows difference only in the constant multipliers.

### 4.3.3 $t_{min}$

The minimum amount of time required to solve a problem instance on a physical machine is non-zero and can be denoted as $t_{min}$. The best case scenario when solving a satisfiable instance is a linear search without backtrack. Depending on the propagator and heuristics used, even such a linear plunge down the search tree can incur a measurable cost. For more realistic instances the likelihood of finding a solution by linear plunge is negligible, and $t_{min}$ can be interpreted to be the amount of time needed before the solution probability becomes significant. The next sub-section looks at how $t_{min}$ is affected when cutoff is measured in units other than time.

The effect of $t_{min}$ was never explicitly analyzed in previous work presented in Chapter 3. But restart strategies can be made more efficient with knowledge of $t_{min}$. It would be possible to avoid restart cutoffs that are smaller than $t_{min}$, as there is no chance of solving the problem and the effort is only wasted. However, such knowledge is often unavailable in practice and the issue is thus frequently overlooked. We now examine the impact of $t_{min}$ on the static restart strategies thus revealing the cost of ignoring this factor in implementation. Specifically, we look at the amount of effort wasted as a function of the total time spent on a strategy. The case for the fixed-cutoff strategy is easily seen to be $t_{min}/t$. The result for Luby's universal strategy is given in Theorem 4.9 and that for the geometric strategy is given in Theorem 4.10.

**Theorem 4.9.** *When applying Luby's universal strategy of length D (i.e. with a deadline of D) to a runtime distribution with $t_{min}$, the fraction of search effort wasted is given by,*

$$\frac{\lceil \log_2 t_{min} \rceil}{\log_2 D} \leq fractionwasted \leq 2\frac{\lceil \log_2 t_{min} \rceil}{\log_2 D}.$$

*Proof.* (Sketch) For a Luby's universal strategy that runs for a duration $D$, the unique number of cutoff values used is at least $\log_2 D$ and at most $\log_2 D/2$. It can be shown that the total time spend on each of these cutoffs is at least $D/\log_2 D$ and at most $2D/\log_2 D$ because of the balanced nature of the strategy. Now, the number of unique cutoff values that are smaller than $t_{min}$ is at most $\lceil \log_2 t_{min} \rceil$. The expression for the fraction of total time wasted on runs that are shorter than $t_{min}$ thus follows. □

**Theorem 4.10.** *When applying the geometric strategy of length D (i.e. with a deadline of D) and scale factor $s = 1$ to a runtime distribution with $t_{min} \geq 1$, the fraction of search effort wasted is given by,*

$$fractionwasted = \frac{r^{\lceil \log_r t_{min} \rceil + 1} - 1}{D(r-1)}.$$

*Proof.* Given a geometric strategy $S = (1, r, r^2, \dots)$, the number of runs wasted is at most $\lceil \log_r t_{min} \rceil$. Thus the upper bound on the amount of time wasted is given by,

$$\sum_{i=0}^{\lceil \log_r t_{min} \rceil} r^i = \frac{r^{\lceil \log_r t_{min} \rceil + 1} - 1}{r-1}.$$

If the strategy runs for length $D$ then the fraction of effort wasted is just,

$$fractionwasted = \frac{r^{\lceil \log_r t_{min} \rceil + 1} - 1}{D(r-1)}.$$

□

### 4.3.4 Other Constraints

The most immediate issue when implementing a restart strategy is in assigning a time unit to the sequence of cutoff values. The two popular time unit choices are the number of seconds and the number of backtracks amongst others given in Section 3.4. When using seconds, the actual runtime is the performance measure of concern which reflects more accurately the perceived effectiveness of the algorithm. The backtrack measure, although sometimes only weakly related to the runtime measure, has the advantage that it implicitly takes $t_{min}$ into consideration. When using backtrack count, the probability of finding a solution is non-zero at *"time"* $= 0$ which is in contrast to the case when time is measured in seconds. However, such probability may be negligible depending on the combination of algorithm and problem instance. Sufficiently significant probability mass may emerge only after a large number of backtracks have been performed. Thus the backtrack measure does not entirely solve the $t_{min}$ problem. There is also the additional disadvantage that improving the backtrack count does not necessarily translate into an improvement in actual runtime.

Another practical constraint is that no matter how difficult, a given problem is always finite and is therefore not a true heavy-tailed distribution in the theoretical sense. The bounded and finite nature of the runtime distribution compromises the notion of asymptotic analysis at the heart of existing results. Chen et al. [6] studied the problem with a model of simple binary search trees and established a correspondence between properties of bounded and unbounded heavy-tailed distributions. They showed that the infinite mean and variance translate into exponentially large mean and variance, and that the finite mean becomes a polynomial mean.

Lastly, a related question is whether the widely accepted heavy-tailed model is really the right choice. Perhaps the heavy-tailed like behaviour can be described equally well with a simpler model. Hoos [25] performed some preliminary experiments in which he found that the runtime distributions can be better modeled as a mixture of generalized exponential distributions. When taking the bounded nature of distributions into consideration, such claim seems particularly plausible. In fact, the Pareto distribution can itself be characterized as a gamma mixture of exponential distributions. Frost et al. [14] on the other hand demonstrated evidence that runtime distributions of solving satisfiable and unsatisfiable random CSPs can be modeled by Weibull and *lognormal* distributions respectively. However, the implication of such alternative modeling on restart strategies, either theoretical or practical, remains an open question.

## 4.4 Summary

In this chapter we presented new results on the study of the fixed-cutoff and the geometric strategies by assuming a simple Pareto runtime distribution. The Pareto distribution belongs to the class of heavy-tailed distributions that have been used successfully in modeling runtime distributions encountered in the real world.

For the fixed-cutoff strategy, we were able to establish a bound on the expected runtime in terms of the Pareto parameter $\alpha$. The interesting observation is that the optimal cutoff remains relatively stable regardless of the value of $\alpha$. The implication is that a fixed-cutoff strategy can potentially be made into a robust static strategy on real world problems.

The geometric strategy, although it can be arbitrarily worse than the optimal fixed-cutoff strategy, is proved to remove heavy tail. The results can be also extended to the more general class of strategies where the geometric factor is a finite polynomial.

We also examined the various practical factors neglected in previous work and noted their impact. When a deadline is imposed, there no longer exists an optimal fixed-cutoff strategy in general. Moreover, there is a disagreement between optimality in expected runtime and optimality in solution probability. The latter is in fact a more suitable performance measure.

We gave expressions showing the performance impact of overhead cost and $t_{min}$. These results would be helpful to practitioners when making trade-off decisions. We also briefly discussed other constraints and issues such as the time unit, the finite nature of runtime distribution and the various possible models for runtime distributions.

# Chapter 5

# New Results on Dynamic Restart Strategies

The series of work by Kautz et al. [26, 30, 40] was innovative. But in spite of an effort at comprehensiveness, there remain areas where results are missing and improvements are possible. In this chapter, we address some of the deficiencies in their work and present a new method of building a runtime classifier that has significantly improved prediction accuracy. We also propose a new hybrid strategy that is shown to be competitive in performance and with enhanced robustness.

## 5.1 Hybrid Geometric-Dynamic Strategy

The foremost deficiency in the work of Kautz et al. [26, 40, 30] was the use of the highly artificial multi-instance setup, introduced in Section 3.4.5, that was favourably biased towards easy instances. In real world situations, it is much more valuable to have an overall performance improvement instead of just a performance improvement on the easier problems. A solver that solves more easy problems more quickly is not as interesting as one that can solve more difficult ones. Instead, our work is aimed at improving the runtime prediction accuracy in the single-instance setup which is also introduced in Section 3.4.5. For this particular setup, Kautz et al. [26] were able to predict whether a run is short, i.e. completes before the median runtime for the dataset, with an accuracy of 60.3%.

Our approach applies similar machine learning techniques but uses a much longer observation horizon in collecting features. Moreover, our feature extraction method is geometrically motivated. This is based on the observation that runs that seem to go on forever often show a stagnating runtime search depth profile. Such profiles have long flat regions indicating very little progress over an extended period of time. The depths oscillate over a narrow range as the solver tries different value assignments for the variables but only finds them infeasible a couple of levels down. Such a behaviour might be attributed to an early mistake in value assignment that has rendered the current subtree unsatisfiable. At the same time, the mistake is subtle enough that the refutation proof cannot be quickly obtained. Figure 5.1 illustrates a typical comparison of a profile that shows stagnation and one that does not.

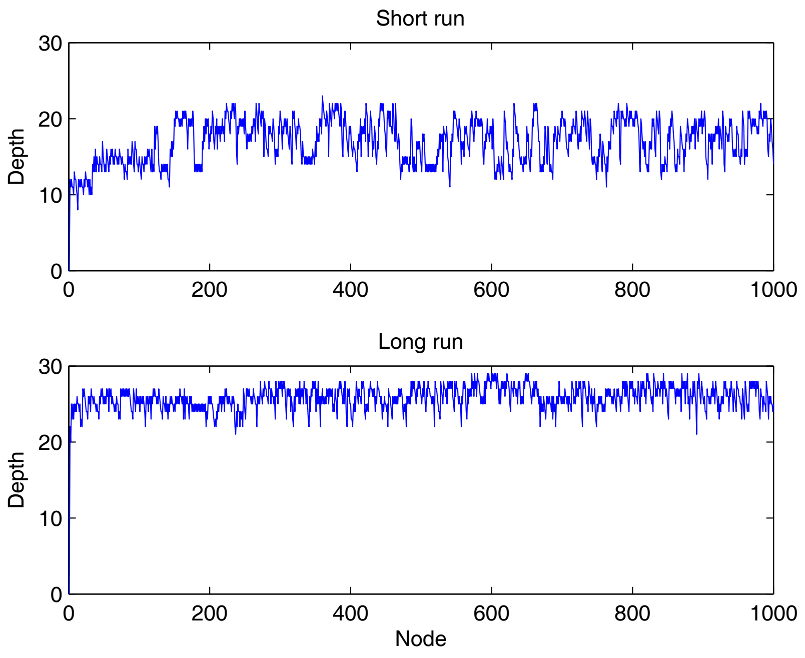

Figure 5.1: Sample depth profiles of short (solved in 1200 seconds) and long runs (timed out in 3600 seconds) for a instruction scheduling problem instance. The instance is given as a set of 68 instructions with constraints such as latency and processor resource. The aim is to schedule the instructions within the given optimal schedule length of 64 time slots.

Based on this observation, it is expected that such a difference in shape can be summarized with suitable metrics and become helpful in distinguishing long and short runs. To make such an approach feasible, the observation horizon needs to be of sufficient length and in our work is set at 50 seconds. Our choice differs qualitatively from the work of Kautz et al. where they had settled on a small observation horizon of 10 search tree nodes [30]. The features collected from such a small horizon is probably inadequate in capturing the characteristics in the search dynamics and is thus more static than dynamic in nature.

Besides helping improve the prediction accuracy, the longer observation horizon also delays the restart decision. This is an undesirable side-effect since many easy problems can be solved with non-trivial probability in time shorter than the observation horizon. For such problems, waiting a long time before making a decision is rather wasteful whereas a few rapid short restarts is much more effective. This is the motivation behind our simple hybrid strategy which starts with a short geometric strategy followed by a dynamic strategy as follows:

$$(sr, sr^2, \ldots, sr^k, d, d, \ldots, d_{last}).$$

In our experiments, we used $s = 1$ and $r = 2$.

The parameter $d$ is the observation horizon length at which point a restart decision is made. If the run is predicted to be long then a restart is performed immediately, otherwise it is allowed to continue until the problem is either solved or the total time allocated to the instance is used up. For easy problems, the geometric strategy portion would ensure efficiency as well as improve the expected runtime. On more difficult problems, i.e. where short runs are generally longer than the observation horizon, the dynamic strategy would help improve the chance of finding a solution.

The last part of our improvement is in the evaluation of performance. Kautz et al. [26, 40, 30] have used expected runtime as the exclusive measure of performance in all of their work. But as mentioned before in Example 4.1, this measure can be misleading when a deadline is imposed, as when solving real problems, because the best expected runtime does not always correspond to the maximum number of problems actually solved. Instead, we evaluate performance both in terms of expected runtime as well as in the number of unique instances solved. This will give a more accurate indication of the practical effectiveness of an algorithm.

## 5.2 Setup for Predictor Inference

A brief overview of the predictor inference workflow was given in Section 2.4 and is restated here for our problem. First, we select as our problem set the QCP, which is defined in the next sub-section. We collect runtime observations for a set of instances from which feature vectors are extracted and filtered. The processed dataset is then split into training and validation data. The training data is used to build a decision tree based predictive model. We adopt an iterative learning procedure using beam search to build the decision tree in which increasing number of features are used. The validation data is then used for selecting the appropriate size of the features set. The learned decision tree is then incorporated into a SAT solver to control when to restart and is evaluated on a separate set of novel instances.

### Problem Set

The problem domain we chose for our study is the quasi-group completion problem (QCP). QCP was first introduced by Gomes and Selman [16] as a crossover between random and structured problems. The synthetic nature of QCP allows a large number of instances to be generated easily and quickly in a controlled fashion so that machine learning techniques can be effectively applied. At the same time, the instances also possess the structured nature of real world problems so the results have practical significance [16].

Graphically, a quasigroup of cardinality $N$ corresponds to an $N \times N$ *Latin square* in which the $N$ symbols are arranged so that each occurs once and only once in each row and each column. A partially filled *Latin square* corresponds to a partial quasigroup. The problem of filling in the blanks to obtain a complete *Latin square* is known as the *quasigroup completion problem*. And like many other benchmark problems, the QCP has been shown to be NP-complete [1, 8].

As in the work of Kautz et al., our study focuses on satisfiable instances. The reason, as previously suggested, is that unsatisfiable problems appear not to have heavy tails [4] and do not benefit from restart strategies. One way to extend the framework to cover all problems would require constructing a satisfiability predictor. But that would be the focus of a separate study.

Satisfiable QCP instances, also known as *quasigroup with holes* (QWH), can be easily created by poking holes in randomly generated complete *Latin squares*. The hardness of the resulting instance is controlled through the size of the problem, the number of holes and whether the distribution of holes is balanced. The problem becomes significantly harder when the number of holes is balanced in the rows and columns [26] and this is the setting we chose to use. Kautz et al. on the other hand had used the easier instances of non-balanced QCPs.

We use the program *lsencode* to create the data set which was developed by Gomes et al. [26] and is available from Gomes's website at *http://www.cs.cornell.edu/gomes/new-demos.htm*. The QCPs generated are of order 34 with 33% holes and are balanced. The dataset files are in the standard *.cnf* format used by most public domain SAT solvers.

## Data Generation

Our SAT solver is a modified version of Satz (version 2.15) developed by Li and Anbulagan [33, 34] and it is one of the fastest SAT solvers. It was also the base solver chosen by Kautz et al. Therefore any performance improvements can be attributed to the restart strategy rather than other algorithmic components.

Modifying the base solver involves several aspects. The first step is in adding randomization to the variable ordering heuristic through random tie-breaking amongst the top 30% high-scorers. This was the setting also used by Kautz et al. [26]. The second aspect is in the recording of runtime observations at each search tree node. We focus on simple measures that are readily available and avoid computationally intensive ones such as the graph constraintness $\lambda$ used by Kautz et al. that measures the constraintness of the binary clause subproblem [26]. The list of measures we record during runtime is given in Table 5.1. We strive for efficiency in the implementation so as to minimize the impact on solver performance. In fact, the processing time taken up by the recording code is always well under a second compared to the minutes and hours required to solve most instances.

| | |
|---|---|
| *lv* | depth of current node |
| *bnd_var* | total number of instantiated variables |
| *pos_vars* | number of variables set to 'true' |
| *clauses* | number of remaining clauses |
| *clause2* | number of remaining binary clauses |
| *score1* | heuristic scores based on look-ahead probing |
| *score2* | another heuristic scores based on look-ahead probing using weighted sum |
| *mono* | number of variables that appear only as positive or only as negative literals |
| *fixed* | number of variables that became instantiated through look-ahead probing |
| *unit* | number of unit propagations performed |

Table 5.1: List of runtime observations recorded.

Lastly, code is added to implement different restart strategies including the fixed-cutoff, the geometric, Luby's universal and the dynamic strategies so that empirical evaluation and comparison can be easily made.

We ran the modified Satz solver with a problem set consisting of 300 instances on a Pentium 4 running at 2.0 GHz with 2 GB of memory. Each instance is repeatedly solved 30 times with different random seeds for a total of 9000 data points. Each run is given a deadline of 3 hours at which point the run is terminated if a solution has not been found.

### Feature Processing

From the basic set of dynamic observations collected, various secondary and composite features are generated. For example, the number of instantiated variables can also be expressed as a fraction of the total number of variables. The latter is probably a more robust measure when problems are of different sizes. Also, we calculate ratios such as the number of variables to the number of active constraints. This measure gives an indication of the constraintness or the difficulty of the subproblems as the search progresses. Another interesting measure is the estimated size of the portion of the complete search tree that has been pruned. There are a total of 22 features after this initial expansion.

The features generated up to this point are given as a collection of time series. For these features to be useful in predictor inference, we need to extract summary statistics for each of these profiles. In addition to the simple *min, max, mean* and *standard deviation* we also include metrics aimed at quantifying the shapes of the time series. For example, we calculate the total spectral power for each of the time series using FFT. We also compute the spectral power in just the lowest 10 frequency components (excluding the zero frequency component). The motivation is again based on the stagnation characteristics mentioned previously. Stagnating profiles often show less oscillation and consequently a lower power signature. At the same time the long flat regions tend to shift the power to the lower frequency domain. We also devise a way to explicitly extract the flat regions from the profiles using a Gaussian filter. Thresholding is used to detect the jumps between regions. An example is shown in Figure 5.2 where jittering is effectively removed by low-pass filtering and only the significant changes remain. The extracted regions are then used to build several metrics including exponentially weighted score on the lengths of the regions, the standard deviation in the region lengths and the total number of regions. In total, there are 207 summary features. Class labels are also assigned to each feature vector separating the solved instances from timed-out instances.

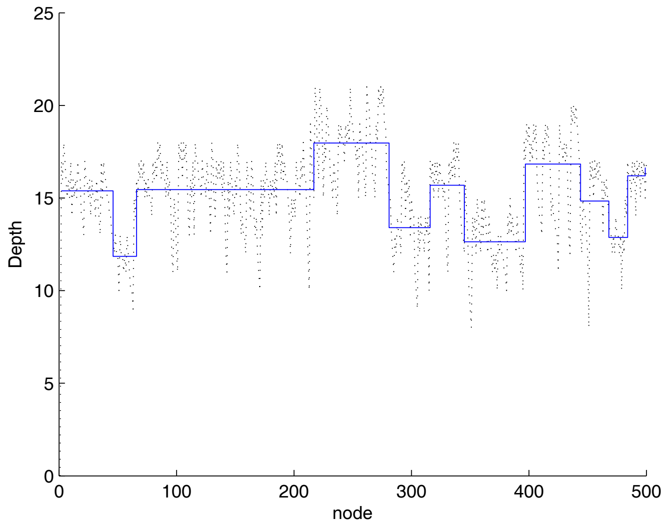

Figure 5.2: Extracting the general shape from time series using Gaussian filtering.

The next step is the filtering of features and removal of the ones deemed redundant or irrelevant. The problem of feature selection and its importance has been widely studied in machine learning and there is a large pool of literature available on the subject (e.g., [21, 29, 31]). We implement a two stage approach for computational efficiency. The first stage involves simple elimination based on correlation and discriminant power. For each pair of features that behave similarly, say with a correlation coefficient of greater than 0.65, only one is kept. Fischer's linear discriminant [11] score, which calculates the ratio of between-class to within-class scatter, is used to decide which one of the two features is kept. The same set of scores is also used to remove features that are really poor discriminators on their own. Depending on the thresholds used, this first stage of feature selection generally removes between 80% to 90% of the features. A sample feature that remains is shown in

Figure 5.3. Observe how it separates the two classes to some degree where the short runs tend to have smaller values for the feature than the timed-out runs.

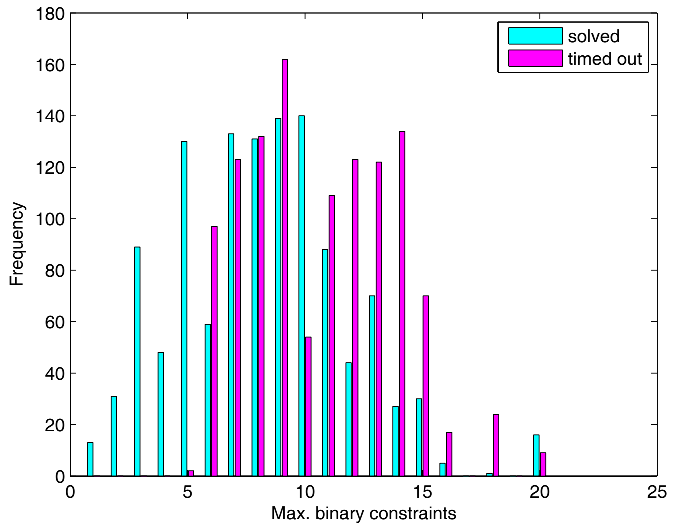

Figure 5.3: A feature that partially discriminates between short and long runs.

## Predictor Learning

It is important to point out that normalization is not done on any of the features. The reason is that when the predictor is applied on novel instances in real-time, the features collected cannot be sensibly normalized without the context of a data set. Thus skipping normalization makes our algorithm applicable in practice.

The second stage of feature selection is integrated into the predictor learning phase where we implement a beam search with a decision tree as the target model [41]. The data is first balanced with down-sampling by randomly removing samples from the class with more samples. The resulting data set has roughly the same number of samples from each class and is then split into 2/3 for the training set and 1/3 for the validation set. During the first epoch, a decision tree is fitted using single features from the feature set and 10-fold cross validation error is calculated for each of the trees. The top 10 features whose trees had the smallest errors are kept. In the second epoch, the features retained from the first epoch are used to form all possible combinations with other original features. A new decisions tree is constructed for each of these 2-feature sets. Thus, the size of the feature subsets are increased by one at each epoch and the process continues for 7 epochs. There is no need to continue further since the error curve is seen to flatten out at epoch 5. The core of our implementation makes use of the tree building functions in *Matlab* running with default parameter settings (i.e., the use of Gini's diversity index in choosing the split).

Although the accuracy of prediction on the training set continues to increase as the feature set is expanded, it is not a good idea to use the largest model due to problems of overfitting and cost consideration. It is more desirable to have a succinct model that offers a good trade-off between performance and complexity. The first step of selection is in deciding on the size of the feature set to use. A plot of the error as a function of feature set size is given in Figure 5.4 that offers useful clues. We find that a feature set of size 3 gives a satisfactory compromise.

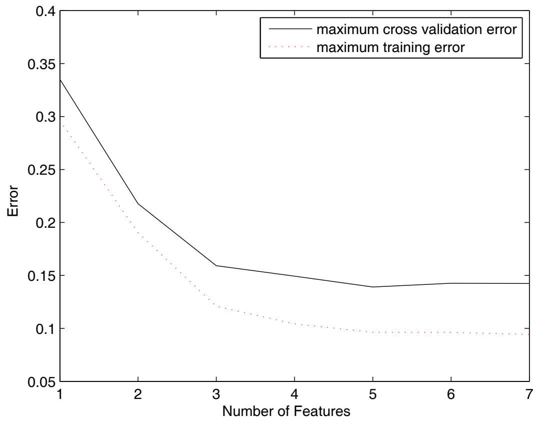

Figure 5.4: Maximum errors as functions of feature set size.

Table 5.2 gives a tally of how frequently each of the features occurs in the 10 feature sets of size 3. It is interesting to observe that some of the most relevant features are based on the raw measures that have been shown to be significant in previous work. For example, Kautz et al. found that the *nogood*, an estimate of the fraction of the complete search tree that has been pruned, is an important feature in their models [26]. Also, the binary constraint metric has always been crucial to SAT solvers and has been used extensively in heuristics and look-ahead algorithms.

Similarly, the 10 decision trees need to be pruned with the same trade-off consideration in mind. Figure 5.5 shows a typical error curve as a function of the number of leaves in the tree. By inspection, a good choice is a tree with 3 leaves. It is worth noting in Figure 5.6 that the timed out instances are predicted much more accurately than the solved instances. Thus, the long runs with the most devastating consequences are effectively removed. This is an important fact that helps ensure the good performance of the restart strategy. An example of the learned decision tree is given in Figure 5.7. The final prediction accuracy on the test set is observed to be between 80% and 90% for the collection of decision trees. This is a significant improvement over the 60.3% accuracy reported by Kautz et al. in the single-instance setup. The improvement can be attributed to a longer observation horizon, a different choice of features in capturing stagnation, as well as a more sophisticated feature selection method. Also, we have used a different dataset consisting of larger and more difficult instances. It should be noted that it is possible that predicting runtime is easier

| | Feature | Frequency |
|---|---|---|
| * | Spectral power of the time series on the number of remaining binary constraints | 10 |
| | Minimum value of the nogood time series | 8 |
| | Mean value of the nogood time series | 2 |
| * | Minimum value of the time series on the % of total future vars instantiated through propagation | 2 |
| * | Low-frequency spectral power of the time series on the number of nogoods | 1 |
| | Minimum value of the search depth time series | 1 |
| * | Low-frequency spectral power of the time series on average score2 | 1 |
| | Maximum value of the time series on the number of remaining binary constraints | 1 |
| | Maximum value of the time series on the number of total remaining constraints | 1 |
| * | Maximum value of the time series on the average variable to constraint count ratio | 1 |
| | Maximum value of the time series on the number of uninstantiated variables | 1 |

Table 5.2: Feature frequencies in the top 10 feature sets of size 3. The features marked with (*) are new features that Kautz et al. have not used. The unmarked ones might have appeared in their work.

on these problems.

## 5.3 Results

To evaluate the performance of our hybrid strategy against static strategies, we generated another 200 QCP problems. When solving these instances, the modified Satz solver first performs a series of geometric restarts until the total time exceeds the given observation horizon. The solver then starts recording runtime observations. At the end of the observation horizon the solver invokes *Matlab* library functions to extract the required summary features and apply the hard-coded predictor to make the restart decision. The procedure typically takes only 1 or 2 seconds and is negligible compared to the observation horizon length. The process is repeated until the problem is solved or the 3 hour deadline is reached. Table 5.3 shows a comparison of performance averaged over 10 trials with different random seeds.

As can be seen, the hybrid strategy is able to solve more instances than Luby's universal strategy, the geometric strategy and even the optimal fixed-cutoff strategy. The hybrid

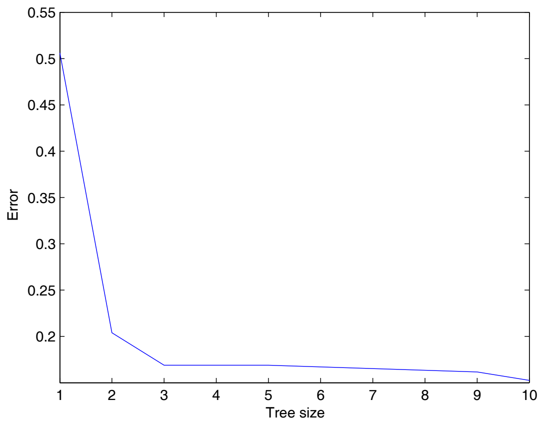

Figure 5.5: Example of cross validation error as a function of tree size.

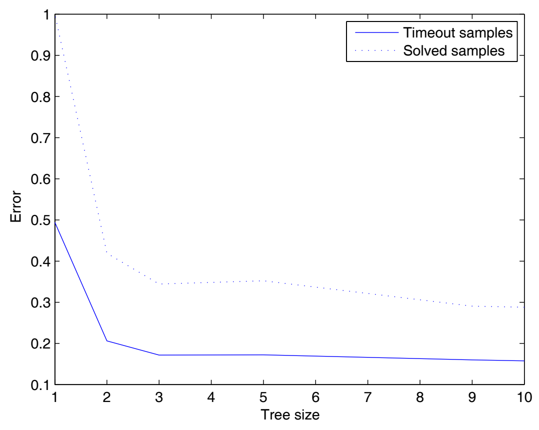

Figure 5.6: Example of cross validation error as a function of tree size for the two different classes of samples.

strategy is also better than a similar strategy that performs fixed-cutoff restart with cutoff 50 after the geometric portion has completed. This illustrates that the performance gain for the hybrid strategy is indeed due to accurate runtime predictions. When comparing the hybrid strategy against a pure dynamic strategy the results are similar. This is expected since it is the dynamic strategy portion that helps improve the chance of finding a solution on problems of intermediate difficulty.

To show the effect of the geometric strategy component, we identified 26 problems that are deemed easy in that they have a nontrivial chance of completing within the observation horizon. Table 5.4 shows a comparison of the expected runtime. It is evident that on easy problems the hybrid strategy outperforms the pure dynamic strategy. This advantage would be significantly amplified on large data sets containing mostly easy problems as is typical in real world situations. Thus the robustness of the hybrid strategy is evident.

To further demonstrate the power of the new runtime predictor we performed a second

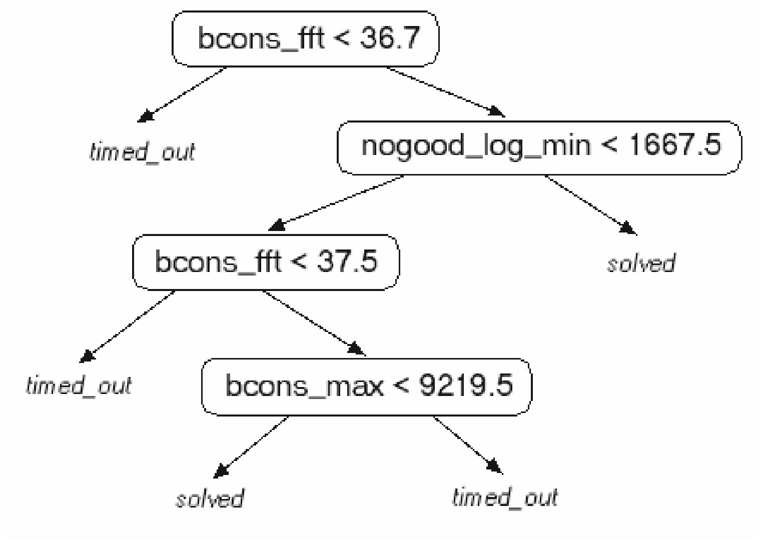

Figure 5.7: One of the learned decision trees.

| | Hybrid | Dynamic | Geometric | Hybrid (Fixed) | Optimal Fixed | Luby |
|---|---|---|---|---|---|---|
| Solved count | 180.5 | 181.0 | 165.5 | 142.3 | 171.0 | 129.4 |
| Mean runtime (hr) | 20.93 | 20.28 | 26.63 | 33.16 | 22.67 | 43.59 |

Table 5.3: Comparative results for various restart strategies averaged over 10 trials. Data set consist of 200 balanced QCP instances of order 34 with 33% holes. The Hybrid and Dynamic strategies use an observation horizon of 50 seconds. The Hybrid (Fixed) is a strategy that performs fixed restart with cutoff 50 after the geometric portion completes. The Hybrid and geometric strategies use a geometric factor of 2 and scale factor 1. Default parameters are used for Luby's universal strategy.

study on a set of much more difficult problems. We generated a set of 100 balanced QCPs of order 37 with 33% holes such that the chance any one of these problems can be solved within the 3 hour deadline is between 3% and 20%. The small size of the dataset is due to the huge increase in computational cost where most of the runs would timeout. Using similar setups as before we are able to obtain a runtime prediction accuracy of around 75%. The decrease in accuracy can be attributed to the smaller data set as well as a particularly short deadline (50 seconds) compared to the average solution time. But when evaluated on a 60 instance test set, the improvement in the number of solved instances is still quite remarkable as shown in Table 5.5.

It is worth noting that we have attempted a similar study using real world scheduling problems. The bottle-neck was in obtaining a large set of homogeneous instances of the right difficulty (i.e. takes on the order of hours to solve). However, focusing on a small set of 4 problems from the *mesa* problem set of the SPEC CPU2000 benchmark, which is a collection of instruction scheduling problems that would be solved within a compiler, where each instance was repeatedly solved 1000 times to generate enough data, we were able to

|  | Hybrid | Dynamic | Geometric | Hybrid (Fixed) | Optimal Fixed | Luby |
|---|---|---|---|---|---|---|
| Mean runtime (s) | 19.0 | 350.8 | 19.4 | 19.3 | 18.3 | 81.2 |

Table 5.4: Comparative results for various restart strategies on 26 easy instances averaged over 10 trials.

|  | Hybrid | Dynamic | Geometric | Hybrid (Fixed) | Optimal Fixed | Luby |
|---|---|---|---|---|---|---|
| Solved count | 36.7 | 38.1 | 14.5 | 5.8 | 18.9 | 6.2 |
| Mean runtime (hr) | 22.90 | 22.21 | 27.59 | 29.16 | 26.41 | 29.58 |

Table 5.5: Comparative results for various restart strategies on hard instances averaged over 10 trials. Data set consists of 60 balanced QCP instances of order 37 with 33% holes.

achieve a prediction accuracy of 95%. Such a high accuracy was due in part to the fact that we used a CSP encoding for the problems. Such an encoding offers much more information than SAT problems. We were able to feasibly compute certain graph based features such as *smallness, eccentricity* and the distribution of *shortest paths*. Thus, with appropriate features and inference techniques, it is promising that dynamic and hybrid strategies can be applied not only to synthetic problems but to real world problems as well. This, however, remains as part of future work.

## 5.4 Summary

The problem settings used in previous studies were scrutinized. We argued that the single-instance setup is more realistic even though it is also more difficult. Moreover, performance measures should be given in terms of the number of problems solved and not just the expected runtime. This is especially important for the multi-instance setup. Thus we tackled the problem of building runtime predictor in the single-instance setup and adopted the more comprehensive evaluation criteria.

We implemented a new inference procedure for building a runtime predictor. The new method differs from existing work in feature collection and selection. Through the use of a longer observation horizon and geometrically-based processing, the features we collected reveal more of the search dynamics. The new feature selection process is more robust by integration with decision tree learning. The result is significantly improved runtime prediction accuracy QCPs compared to previous work.

We further improved solver robustness over pure dynamic strategy by combining it with a geometric strategy. The resulting hybrid strategy showed good performance on both easy and hard problems and far surpasses the existing static strategies.

# Chapter 6

# Conclusions

This chapter summarizes our contributions to the static and dynamic restart strategies. We also briefly offer some recommendations for future work.

## Contributions

A wide variety of physical problems can be modeled as CSP and SAT instances. The resulting combinatorial problems can be solved by systematic backtrack search but there is often a great variation in runtime performance. The difficulty lies in the heuristics for making branching decisions and a mistake made near the top of the search tree can result in an exponential increase in search effort.

Randomization and restart is an effective technique for alleviating the impact of bad heuristic decisions, particularly on problems that suffer from heavy tails. Under special ideal conditions, a simple fixed-cutoff strategy can be made optimal and Luby's universal strategy has the best worst-case performance on general problems [35]. However, Luby's universal strategy has been shown in empirical studies to converge slowly [19, 30, 40]. The geometric strategy [45] is an answer to the slow convergence of Luby's strategy and has shown good empirical performance [45, 47].

The runtime distributions encountered when solving CSP problems are in fact not arbitrary, but can be reasonably modeled with a heavy-tail distribution. Thus, we analyzed the fixed-cutoff and geometric strategies on the Pareto distributions. For the fixed-cutoff strategy we found that the optimal cutoff is stable regardless of the heaviness of the tail. For the geometric strategy, we proved for the first time that it removes heavy tail. However, we also showed that for general runtime distributions, the worst-case performance of the geometric strategy is not bounded.

We also examined the effects of various factors encountered in applying restart techniques in practice that have been overlooked in previous work. We gave an expression for the expected performance of the fixed-cutoff strategy when a deadline is imposed and found that it is no longer optimal. Moreover, the strategy that minimizes the expected runtime does not always maximize the solution probability. We argued that the latter is in fact a more relevant measure of performance. We also gave an expression showing the effect of restart overhead on the expected runtime of the fixed-cutoff strategy and observed that it has a similar weight as the fixed cutoff used. Lastly, results were given for Luby's universal and geometric strategies quantifying the effort wasted when the starting cutoff is smaller than $t_{min}$, where $t_{min}$ is the minimum amount of time needed in finding a solution.

The dynamic restart strategy is a more sophisticated restart technique which applies machine learning techniques to build runtime predictors from runtime evidence. The runtime predictions are then used to guide restart decisions. The current state-of-the-art dynamic restart strategy uses a short observation horizon and simple features. The performance can be poor in a single-instance setup. The performance in the multi-instance scenario is much better but is marred by the pitfall that the difficult instances are simply skipped.

We extended the dynamic restart technique to the single-instance setup. The new procedure for building a runtime predictor uses a longer observation horizon and geometrically based features. We also performed iterative feature selection during inference. The resulting improvement in performance is significant. However, in a practical application, non-trivial effort is needed in collecting runtime features and making predictions. This becomes a concern for easy problems that can be much more quickly solved by a few rapid short restarts. Thus we proposed a hybrid strategy that combines the geometric and dynamic strategies. The new strategy is robust on both easy and more difficult instances compared to pure dynamic strategy. We compared the performance of various strategies on quasi-group completion problems both in terms of expected runtime and, more relevantly, on the number of instances solved. The new strategy is able to improve the expected runtime on easy instances and solution probability on hard instances. The hybrid strategy is also shown to out-perform the optimal fixed-cutoff strategy.

## Future Work

Static strategies are easy to implement and will probably continue to see wide usage thus justifying further research. From a theoretical perspective, one interesting open question is whether the geometric strategy removes the heavy tail for arbitrary runtime distributions. A positive result would significantly boost practitioners' confidence in applying a geometric strategy which has already demonstrated good empirical performance. A related question is whether the performance of a geometric strategy with fixed parameters can be bounded with respect to that of the optimal fixed-cutoff strategy on the Pareto distribution.

Further analysis of various restart strategies on other relevant distributions, i.e. ones which can be used to model real world runtime distributions, would also be useful in offering interesting insights. Some candidates include the gamma distribution and mixed generalized exponential distributions. Recall that an interesting fact we learned for the Pareto distribution is the relatively constant optimal fixed cutoff. If inference techniques can be used to guess the correct time unit then we may be able to effectively apply the fixed-cutoff strategy in practice. It is also plausible to infer $t_{min}$ which will help improve performance of static strategies on real world problems.

A more sophisticated research direction involving machine learning is in examining the consistencies or lack thereof in runtime distributions within and across problem subclasses. We would like to associate with each type of runtime distribution a restart strategy that is particularly effective, possibly because of some special features of the distribution. We will then be able to apply instance specific static strategies. A special case would be to build a satisfiable/unsatisfiable predictor and avoid restarts on problems that are unlikely to have solutions.

For the dynamic strategies, which already outperform the static ones, there are also several possible improvements. For example, a confidence measure can be associated with each runtime prediction. When it is below some threshold, we would like to delay the restart decision until further evidence can be gathered. Also, a hierarchical runtime predictor can be constructed in which features are introduced successively to improve the prediction confidence but take more and more computational resource to compute. For example, efficiency would be improved if one can avoid a large number of FFT operations and use mainly *min* and *max*.

Lastly, studies on the dynamic strategies should be expanded to real world problem sets, particularly that of CSPs. Our preliminary results suggest that the performance can be greatly improved since the CSP encoding allows more sophisticated features that are unavailable or impractical for SAT. For example, distributions of shortest path metrics in the constraint graph and distributions of domain sizes have been found to be very prominent features on scheduling problems. The issue in using real world problems is that they are exceedingly heterogeneous, and there are often only a few instances for each subclass of sufficient homogeneity.

# Bibliography

[1] D. Achlioptas, C. P. Gomes, H. Kautz, and B. Selman. Generating satisfiable problem instances. In *Proceedings of the* $17^{th}$ *National Conference on Artificial Intelligence*, pages 256–261, 2000.

[2] H. Alt, L. J. Guibas, K. Mehlhorn, R. M. Karp, and A. Wigderson. A method for obtaining randomized algorithms with small tail probabilities. Research Report MPI-I-92-110, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, March 1992.

[3] H. Alt, L. J. Guibas, K. Mehlhorn, R. M. Karp, and A. Wigderson. A method for obtaining randomized algorithms with small tail probabilities. *Algorithmica*, 16(4/5):543–547, 1996.

[4] C. Bessiere, C. Fernandez, C. P. Gomes, and B. Selman. To be or not to be heavy-tailed. *Unpublished manuscript*, 2003.

[5] J. R. Bitner and E. M. Reingold. Backtrack programming techniques. *Communications of the ACM*, 18:651–656, 1975.

[6] H. Chen, C. P. Gomes, and B. Selman. Formal models of heavy-tailed behavior in combinatorial search. In *Proceedings of the* $7^{th}$ *International Conference on Principles and Practice of Constraint Programming*, pages 408–421, 2001.

[7] X. Chen. *A theoretical comparison of selected CSP solving and modeling techniques.* PhD thesis, University of Alberta, Department of Computing Science, 2000.

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

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

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

[11] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification, 2nd Ed.* John Wiley & Sons, Inc., New York, 2001.

[12] J. W. Freeman. *Improvements to propositional satisfiability search algorithms.* PhD thesis, University of Pennsylvania, 1995.

[13] A. M. Frisch and Y. Zhan. Restart strategies for constraint satisfaction problems. *Automated Reasoning Workshop*, pages 67–100, 2001.

[14] D. Frost, I. Rish, and L. Vila. Summarizing CSP hardness with continuous probability distributions. In *Proceedings of the* $14^{th}$ *National Conference on Artificial Intelligence*, pages 327–333, 1997.

[15] C. P. Gomes, C. Fernandez, B. Selman, and C. Bessiere. Statistical regimes across constrainedness regions. *Constraints*, 10:317–337, 2005.

[16] C. P. Gomes and B. Selman. Problem structure in the presence of perturbations. In *Proceedings of the* $14^{th}$ *National Conference on Artificial Intelligence*, pages 221–226, 1997.

[17] C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126:43–62, 2001.

[18] C. P. Gomes, B. Selman, and N. Crato. Heavy-tailed distributions in combinatorial search. In *Proceedings of the* $3^{rd}$ *International Conference on Principles and Practice of Constraint Programming*, pages 121–135, 1997.

[19] C. P. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomenon in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24:67–100, 2000.

[20] C. P. Gomes, B. Selman, and H. Kautz. Boosting combinatorial search through randomization. In *Proceedings of the* $15^{th}$ *National Conference on Artificial Intelligence*, pages 431–437, Madison, Wisconsin, 1998.

[21] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[22] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[23] W. D. Harvey. *Nonsystematic backtracking search.* PhD thesis, Standford University, 1995.

[24] T. Hogg and C. P. Williams. Solving the really hard problems with cooperative search. In *Proceedings of the* $11^{th}$ *National Conference on Artificial Intelligence*, pages 231–236, 1993.

[25] H. Hoos. Heavy-tailed behaviour in randomised systematic search algorithms for SAT. Technical Report TR-99-160, Computer Science Department, The University of British Columbia, 1999.

[26] E. Horvitz, Y. Ruan, C. P. Gomes, H. Kautz, B. Selman, and D. M. Chickering. A bayesian approach to tackling hard computational problems. In *Proceedings of the* $17^{th}$ *Conference on Uncertainty and Artificial Intelligence*, pages 235–244, 2001.

[27] B. A. Huberman, R. M. Lukose, and T. Hogg. An economic approach to hard computational problems. *Science*, 27:51–53, 1997.

[28] T. Hulubei and B. O'Sullivan. Search heuristics and heavy-tailed behaviour. In *Proceedings of the* $11^{th}$ *International Conference on Principles and Practice of Constraint Programming*, pages 328–342, Stiges, Spain, 2005.

[29] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *International Conference on Machine Learning*, pages 121–129, 1994.

[30] H. Kautz, E. Horvitz, Y. Ruan, C. P. Gomes, and B. Selman. Dynamic restart policies. In *Proceedings of the* $18^{th}$ *National Conference on Artificial Intelligence*, pages 674–681, 2002.

[31] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[32] K. Leyton-Brown, E. Nudelman, and Y. Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Proceedings of the* $8^{th}$ *International Conference on Theory and Practice of Constraint Programming*, pages 556–572, 2002.

[33] C. M. Li. A constraint-based approach to narrow search trees for satisfiability. *Information Processing Letters*, 71(2):75–80, 1999.

[34] C. M. Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the* $15^{th}$ *International Joint Conference on Artificial Intelligence*, pages 366–371, 1997.

[35] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47:173–180, 1993.

[36] B. B. Mandelbrot. The Pareto-Levy law and the distribution of income. *International Economic Review*, 1:79–106, 1960.

[37] B. B. Mandelbrot. *The fractal geometry of nature.* Freeman: New York, 1983.

[38] B. Ó Nualláin, M. de Rijke, and J. van Benthem. Ensemble-based prediction of SAT search behaviour. *Electronic Notes in Discrete Mathematics*, 9, 2001.

[39] Y. Ruan. Hardness-aware restart policies. $18^{th}$ International Joint Conference on Artificial Intelligence: Workshop on Stochastic Search, 2003.

[40] Y. Ruan, E. Horvitz, and H. Kautz. Restart policies with dependence among runs: A dynamic programming approach. *Proceedings of the* $8^{th}$ *International Conference on Principles and Practice of Constraint Programming*, pages 573–586, 2002.

[41] T. Russel, A. M. Malik, M. Chase, and P. van Beek. Learning basic block scheduling heuristics from optimal data. In *Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research*, pages 242–253. IBM Press, 2005.

[42] E. P. K. Tsang. *Foundations of Constraint Satisfaction.* Academic Press, 1993.

[43] A. van Moorsel and K. Wolter. Analysis and algorithms for restart. In *Proceedings of the* $1^{st}$ *International Conference on the Quantitative Evaluation of Systems*, pages 195–204, Washington, DC, USA, 2004. IEEE Computer Society.

[44] A. van Moorsel and K. Wolter. Meeting deadlines through restart. In *Proceedings of the* $12^{th}$ *GI/ITG Conference on Measuring, Modeling and Evaluation of Computer and Communication Systems*, pages 155–160, Dresden, Germany, 2004.

[45] T. Walsh. Search in a small world. In *Proceedings of the* $16^{th}$ *International Joint Conference on Artificial Intelligence*, pages 1172–1177, 1999.

[46] R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of the* $18^{th}$ *International Joint Conference on Artificial Intelligence*, pages 698–701, 2003.

[47] Y. Zhan. Randomization and restarts on a state of the art SAT solver - SATZ. Master's thesis, University of York, Department of Computing Science, 2001.

[48] H. Zhang. A random jump strategy for combinatorial search. In *Proceedings of the* $7^{th}$ *International Symposium on Artificial Intelligence and Mathematics*, 2002.