On Universal Restart Strategies for Backtracking Search

Huayue Wu and Peter van Beek

School of Computer Science, University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
{hwu,vanbeek}@cs.uwaterloo.ca

Abstract. Constraint satisfaction and propositional satisfiability problems are often solved using backtracking search. Previous studies have shown that a technique called randomization and restarts can dramatically improve the performance of a backtracking algorithm on some instances. We consider the commonly occurring scenario where one is to solve an ensemble of instances using a backtracking algorithm and wish to learn a good restart strategy for the ensemble. In contrast to much previous work, our focus is on universal strategies. We contribute to the theoretical understanding of universal strategies and demonstrate both analytically and empirically the pitfalls of non-universal strategies. We also propose a simple approach for learning good universal restart strategies and demonstrate the effectiveness and robustness of our approach through an extensive empirical evaluation on a real-world testbed.

1 Introduction

Constraint satisfaction and propositional satisfiability problems are often solved using backtracking search. It has been widely observed that backtracking algorithms can be brittle on some instances. Seemingly small changes to a variable or value ordering heuristic, such as a change in the ordering of tie-breaking schemes, can lead to great differences in running time. An explanation for this phenomenon is that ordering heuristics make mistakes. Depending on the number of mistakes and how early in the search the mistakes are made (and therefore how costly they may be to correct), there can be a large variability in performance between different heuristics. A technique called randomization and restarts has been proposed for taking advantage of this variability [1–4].

A restart strategy $(t_1, t_2, t_3, ...)$ is an infinite sequence where each t_i is either a positive integer or infinity. The idea is that a randomized backtracking algorithm is run for t_1 steps. If no solution is found within that cutoff, the algorithm is run for t_2 steps, and so on. The usual method for randomizing a backtracking algorithm is to randomize the variable or value ordering heuristics (e.g., [2, 4]).

Luby, Sinclair, and Zuckerman [1] examine restart strategies in the more general setting of Las Vegas algorithms. A Las Vegas algorithm is a randomized algorithm that always gives the correct answer when it terminates, however the running time of the algorithm varies from one run to another and can be modeled

as a random variable. Let $f(t)$ be the probability that a backtracking algorithm $\mathcal{A}$ applied to instance x stops after taking exactly t steps; $f(t)$ is referred to as the runtime distribution of algorithm $\mathcal{A}$ on instance x. Luby, Sinclair, and Zuckerman [1] show that, given full knowledge of the runtime distribution of an instance, the optimal strategy for that instance is given by $(t^*, t^*, t^*, \ldots)$, for some fixed cutoff t^*. Of course, the runtime distribution of an instance is not known in practice.

A fixed cutoff strategy is an example of a non-universal strategy: designed to work on a particular instance—or, more precisely, a particular runtime distribution. When applied to another instance for which it was not designed, non-universal strategies are open to catastrophic failure, where the strategy provably will fail on the instance no matter how much time is alloted to the backtracking search. The failure is due to all cutoffs being too small, before there is sufficient probability of solving the instance. To avoid failure, such restart strategies are sometimes set with cutoffs much too high with a serious consequent hit in performance (see, e.g., Huang [5] and references therein).

In contrast to non-universal strategies, universal strategies are designed to be used on any instance. Luby, Sinclair, and Zuckerman [1] were the first to propose a universal strategy (hereafter, the Luby strategy). The Luby strategy is given by $(1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, 1, \ldots)$ and grows linearly (each t_i is bounded above by $(i+1)/2$). Walsh [6] proposes the universal strategy $(1, r, r^2, \ldots)$, where the restart values are geometrically increasing. In practice, it has been found that a geometric factor in the range $1 < r < 2$ often works well on both SAT and CSP instances [6, 7].

We consider a setting where an ensemble or sequence of instances are to be solved over time. Such a setting often arises in practice. For example, a common scenario in scheduling and rostering is that at regular intervals on the calendar a similar scheduling problem must be solved. For a further example, in our evaluation testbed of instruction scheduling, thousands of instances arise each time a compiler is invoked on some software project. In such a setting, the question we address is whether we can learn a good restart strategy in an offline manner from a small sample of the instances. In contrast to previous work, our focus is on learning good universal strategies.

In this paper, we make the following three contributions. First, we demonstrate both analytically and empirically the pitfalls of non-universal strategies, as learned by previously proposed approaches.

Second, we examine the worst-case performance of the universal strategies. The Luby universal strategy is known to be within a log factor of optimal in the worst-case [1]. However, bounds on the worst-case performance of the Walsh universal strategy are not known. We show that the performance of the Walsh strategy is not bounded with respect to the optimal value in the worst-case. The proof of the theorem provides some intuition behind good choices for the geometric factor that have previously been determined empirically. We also prove that the Walsh strategy guarantees a performance improvement under certain conditions.

Finally, we examine the practical performance of the universal strategies. Previous empirical evaluations have reported that the Luby universal strategy can perform poorly in practice (e.g., [4, 8–10]) and the Walsh strategy has not been thoroughly evaluated empirically. We show that the performance of the universal strategies can sometimes be considerably improved by parameterizing the strategies and estimating the optimal settings for these parameters from a small sample of instances. The two parameters that we consider are a scale parameter s and a geometric factor parameter r. The Walsh strategy already contains the geometric factor. Luby, Sinclair, and Zuckerman [1, p.179] note that a geometric factor can also be incorporated into their universal strategy. For example, the first few terms of the Luby strategy with a geometric factor of 3 are given by, 1, 1, 1, 3, 1, 1, 1, 3, 1, 1, 1, 3, 9, ... The scale parameter, as the name suggests, scales, or multiplies, each cutoff in a restart strategy. For example, the first few terms of a scaled Walsh strategy are given by $(s, sr, sr^2, \ldots)$, for some given scale s and geometric factor r. Parameterizing the strategies improves performance while retaining any optimality and worst-case guarantees. We demonstrate the effectiveness and robustness of our approach through an extensive empirical evaluation on a real-world testbed of instruction scheduling problems.

2 Related Work

In this section, we relate our work to previously proposed methodologies for learning good restart strategies for an ensemble of instances. In each methodology, one begins by choosing a sample of instances from the ensemble.

Restart strategies are often informally chosen using trial-and-error methods where one runs experiments on a sample of instances in order to find good strategies (e.g., [4, 11]). Zhan [7] performs extensive experiments to evaluate which geometric factor r for the universal strategies works best across problem classes. In preliminary work, we suggested that the universal strategies could be further parameterized using a scale factor s and that the parameters could be tuned to improve performance [12]. Independently, Eén and Sörensson [13] incorporates a scaled ($s = 150$) Walsh strategy into their SAT solver, but they offer no justification for the choice of value. More recently, Huang [5] extensively compares fixed cutoff restart strategies with particular scaled Luby and Walsh strategies on SAT benchmarks in the context of conflict clause learning. Huang notes that none of the strategies evaluated was consistently best across all benchmark families, which suggests that adapting a strategy to a benchmark family is important.

More formal methods for choosing good restart strategies have also been proposed. Ó Nualláin, de Rijke, and van Benthem [14] consider the case where an ensemble consists of instances drawn from two known runtime distributions—a runtime distribution for satisfiable instances and one for unsatisfiable—but the runtime distribution for any given instance is unknown. They sketch how esti-

mates of these runtime distributions can be used to derive good restart strategies. However, the work is preliminary and no experimental results are reported.

Kautz et al. [8, 9] consider the case where an ensemble consists of instances drawn from n different runtime distributions. They show that with this additional information, one can use dynamic programming to derive good restart strategies and that these restart strategies can be further improved by incorporating observations from during the search process itself. However, their approach is developed and experimentally evaluated in a scenario where a *new* instance is chosen each time the backtracking algorithm is restarted. Unfortunately, this is not a realistic scenario. In practice, one is interested in solving each instance in the ensemble, not just any instance.

Ruan, Horvitz, and Kautz [10] determine a good restart strategy as follows. One first empirically constructs the runtime distributions for each of the instances in the sample. The instances are then clustered such that the runtime distributions of the instances in a cluster have low variability. Each cluster of runtime distributions yields a sub-ensemble runtime distribution which is the normalized sum of the runtime distributions of the instances it contains. These sub-ensemble runtime distributions are then used to construct a single, final strategy using dynamic programming. However, the resulting learned strategies are non-universal and thus are open to catastrophic failure.

Gagliolo and Schmidhuber [15] propose learning a good restart strategy online as instances of the ensemble are solved. The restart strategy learned is an interleaving of the Luby universal strategy and a fixed cutoff. In essence, the fixed cutoff is determined by constructing a single runtime distribution from the samples seen so far, and choosing a restart strategy that minimizes the expected runtime on that runtime distribution. Because of the interleaved Luby strategy, the learned strategy is not open to catastrophic failure. However, we argue (see subsequent sections) that the fixed cutoff learned in this approach may not be useful in many practical settings.

3 Theoretical Results

In this section we contribute to the theoretical understanding of the universal restart strategies. For universal strategies there are two worst-case bounds of interest: worst-case bounds on the expected runtime of a strategy and worst-case bounds on the tail probability of a strategy. The tail probability is the probability that the strategy runs for more than t steps, for some t.

3.1 Bounds on Expected Runtime

Luby, Sinclair, and Zuckerman [1] show that, for any runtime distribution, the Luby strategy is within a log factor of the optimal fixed cutoff strategy for that distribution. They also show that, given no knowledge of the runtime distribution, no universal strategy can have a better bound than being within a log factor of optimal. The question we address is, how far can the Walsh strategy

be from the optimal fixed cutoff strategy for a runtime distribution? Does it attain the best-possible log factor bound and, if not, can it be bound by some other polynomial or exponential function? Unfortunately, the answer is no to all of these questions. The first notable property we establish is that the Walsh strategy, although it appears to work well in practice, unfortunately comes with no formal guarantee on its worst-case performance as it can be *unbounded* worse than the optimal fixed cutoff strategy.

Theorem 1. *The expected runtime of a Walsh strategy of the form,* $(1, r, r^2, \ldots)$*,* $r > 1$*, can be unbounded worse than that of the optimal fixed cutoff strategy.*

Proof. The proof is constructive; given a Walsh strategy, we give a method for constructing a runtime distribution such that the expected runtime of the Walsh strategy on the runtime distribution is unbounded. Let $\mathcal{A}$ be the algorithm being applied by the strategy. For any given Walsh strategy of the form $(1, r, r^2, \ldots)$, $r > 1$, define a runtime probability distribution for $\mathcal{A}$ as follows,[1]

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

where $l = r/(r-1)$. Note that the optimal strategy for this runtime distribution is the fixed cutoff strategy $(1, 1, 1, \ldots)$ with expected runtime l. The expected runtime $E[T]$ of the Walsh strategy is given by,

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

Thus the expected runtime $E[T]$ is unbounded with respect to that of the optimal fixed cutoff strategy. □

[1] Note that the condition $t = \infty$ in the runtime probability distribution simplifies the proof. However, for real CSP and SAT instances there always exists a t for which $P(T \leq t) = 1$; i.e. the runtime distributions are finite as a backtracking algorithm will always (eventually) terminate. At the expense of complicating the proof, the condition $t = \infty$ can also be formulated as $t = r^k$ and it can be shown that the Walsh strategy can be exponentially worse than the optimal fixed cutoff strategy.

The proof of the theorem provides some practical guidance for selecting a value for the geometric factor r. Values in the range $1 < r < 2$ are safer, and the smaller the probability of a short run, the closer the geometric factor r must be to 1, or the strategy may not converge.

3.2 Bounds on Tail Probability

Recall that the tail probability of an algorithm $\mathcal{A}$ on some instance is the probability that the algorithm runs for more than t steps on that instance, for some t. Similarly, the tail probability of a restart strategy on some instance is the probability that the strategy runs for more than t steps. Luby, Sinclair, and Zuckerman [1] show that, no matter what the runtime distribution of the original algorithm $\mathcal{A}$, if we apply $\mathcal{A}$ using the Luby strategy, the tail probability of the restart strategy decays superpolynomially as a function of t (i.e., faster than polynomially). Here we establish a more restricted result for the Walsh strategy by focusing on heavy-tailed probability distributions of the Pareto-Lévy form. Gomes et al. [3, 4] show that this family of distributions can be a good fit to the runtime distributions of backtracking algorithms with randomized heuristics in the cases where restarts improve performance the most. In Pareto distributions, the tail probability has the form,

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

In contrast to the classic exponentially decaying tail, a heavy-tailed distribution is one where the tail probability decays polynomially. In terms of a backtracking algorithm, a heavy-tail model implies that there is a significant probability that the backtracking search will run for a long time, and the longer the backtracking search has been running, the longer additional time it can be expected to run.

Theorem 2. *If the runtime distribution of the original algorithm $\mathcal{A}$ is a heavy-tailed probability distribution of the Pareto-Lévy form and $\mathcal{A}$ is applied using a Walsh strategy of the form, $(1, r, r^2, \ldots)$, $r > 1$, the tail probability of the restart strategy decays superpolynomially.*

Proof. Let $F(t) = 1 - Ct^{-\alpha}$ be the cumulative distribution of the runtime of algorithm $\mathcal{A}$; i.e., the probability that $\mathcal{A}$ stops after taking t or fewer steps. The idea is to first bound the tail probability of the restart strategy from above by a piecewise linear function which is further bounded by a smooth function that decays faster than heavy tail. Let $i_t = \arg\max_i(r^i \leq t) = \lfloor \log_r t \rfloor$.

$$\begin{aligned} P[T > t] &\leq P[T > r^{i_t}] \\ &= \prod_{l=0}^{i_t} \left(1 - F\left(r^l\right)\right) \\ &= C^{\lfloor \log_r t \rfloor + 1} \; r^{-\alpha \sum_{l=0}^{\lfloor \log_r t \rfloor} l} \end{aligned}$$

Since $\sum_{l=0}^{\lfloor \log_r t \rfloor} l = \frac{1}{2}(1 + \lfloor \log_r t \rfloor)\lfloor \log_r t \rfloor \geq \frac{1}{2}(1 + (\log_r t - 1))(\log_r t - 1) = \frac{1}{2}(\log_r^2 t - \log_r t)$, together with the fact that $r > 1$, we have,

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

In the case where $C < 1$, we have,

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

where $R = \frac{\alpha}{2} + \log_r C$. In the case where $C \geq 1$, we have,

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

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

A practical consequence of the theorem is that for instances for which a Pareto distribution is a sufficiently good fit, the Walsh strategy is guaranteed to reduce the probability of very long runs. Moreover, it is quick to check that both the mean and the variance are finite for the cumulative distribution $P[T \leq t] = 1 - e^{-\log^2 t}$, and thus the mean and variance of the Walsh strategy are also finite. In contrast, the mean of the Pareto distribution is unbounded if $\alpha \leq 1$ and the variance is unbounded if $\alpha \leq 2$. Thus, for certain settings of the parameter α in the Pareto distribution, the Walsh strategy is guaranteed to give an expected performance gain over not performing restarts.

4 Analytical Study

In this section we use a simple analytic example to illustrate the pitfalls of using fixed cutoff and other non-universal strategies on an ensemble of instances. For the purposes of this study, we assume that there is no limit on computational resources—i.e., we wish to run the strategy on each instance in the ensemble until a solution is found. In such a scenario, the appropriate performance measure is the expected cost of solving the ensemble.

Consider an ensemble of n instances where the (unknown) runtime distribution of the k^{th} instance is given by $f_k(t)$, $k = 0, \ldots, n-1$,

$$f_k(t) = \begin{cases} p & t = 10^k \\ 1 - p & t = 10^n \\ 0 & \text{otherwise.} \end{cases}$$

where $p \geq \frac{1}{8}$ (the lower bound on p is to ensure that restarts are helpful for all possible values of n and k).

Following Gagliolo and Schmidhuber [15], suppose that we learn a fixed cutoff by collecting a sample of the ensemble, constructing a single runtime distribution for the sample, and finally choosing the fixed cutoff that minimizes the expected runtime of the strategy given the runtime distribution. In the limiting case where the sample is the entire ensemble, the runtime distribution would be given by,

$$f(t) = \begin{cases} p/n & t = 10^k, k = 0, \ldots, n-1 \\ 1-p & t = 10^n \\ 0 & \text{otherwise.} \end{cases}$$

It can easily be shown that (1, 1, ...) is the optimal fixed cutoff strategy for the above runtime distribution. However, if we actually run the strategy on each instance of the ensemble, the expected runtime of solving the ensemble is *unbounded* (i.e., the process will not terminate). This is true for any fixed cutoff strategy (t, t, ...) where $t < 10^{n-1}$. Further, if we interleave the learned cutoff with the Luby strategy, the result would only degrade the performance of the Luby.

Following Ruan, Horvitz, and Kautz [10], suppose that we learn a cutoff strategy by constructing a runtime distribution for each instance in the sample, clustering the runtime distributions into a small number of classes, and learning a restart strategy from the clustered runtime distributions. It can be shown that, under reasonable assumptions about the method for clustering the runtime distributions, unless the sample consists of the entire ensemble and the number of clusters is equal to n, the expected runtime of the learned strategy on the ensemble of instances can be unbounded.

As a point of comparison, it is easy to show that for each instance k with runtime distribution $f_k(t)$, $k = 0, \ldots, n-1$, the optimal restart strategy for that instance is the fixed cutoff strategy (t_k^*, t_k^*, ...) with $t_k^* = 10^k$. For the ensemble of n instances, if we use the optimal restart strategy for each instance (i.e., the fixed cutoff strategy (t_k^*, t_k^*, ...) is used on instance k), the expected runtime of solving the ensemble is $(10^n - 1)/(9p)$. Similarly, the expected runtime of the strategy using the cutoff $t = \infty$ (i.e., running the backtracking algorithm to completion on each instance) can be derived and is obviously finite. Thus, the non-universal restart strategies learned by previous proposals can be unbounded worse than the gold standard strategy of using the fixed cutoff t_k^* and unbounded worse than performing no restarts at all.

The universal strategies, by design, avoid this pitfall. But the question now is, how well do they perform? For a given n and p, we can experimentally determine the expected runtime of the Luby and Walsh strategies on the ensemble of instances, for various parameter settings. Table 1 summarizes a representative example, the case where $n = 6$ and $p = 1/8$. Ratios less then one in the table represent a speed-up; greater than one represent a slowdown. The default value of the scale parameter s is 1 and the default value of the geometric parameter r is 2 for the Luby strategy and 1.1 for the Walsh strategy.

There are two aspects of this analytic example that are worth noting. First, the pitfall that a learned non-universal restart strategy can be arbitrarily worse

Table 1. Ratio of expected runtime of the Luby and Walsh restart strategies over (a) the expected runtime of the gold standard strategy t_k^*, and (b) the expected runtime when no restarts are performed.

	restarts using t_k^*		no restarts	
Parameter settings	Luby	Walsh	Luby	Walsh
Optimal	2.51	3.26	0.42	0.55
Default	14.17	3.30	2.39	0.56

than no restarting at all arose even when perfect information about the runtime distributions was available (i.e., the sample consisted of the entire ensemble). The pitfall is exacerbated when only estimates are available. In particular, the pitfall would be likely to arise whenever there exist instances in the ensemble that are inherently harder to solve than others. We would argue that these conditions would often hold in practice, an assertion that our empirical study in the next section supports. Second, the universal strategies avoid the pitfall and offer speedups for various parameter settings. In the next section, we introduce more realistic runtime distributions and study how robustly and accurately good parameter settings can be estimated and just how much the parameterized universal strategies can lead to performance gains.

5 Empirical Study

In this section we present the results of an extensive empirical evaluation of our approach on real-world scheduling instances. We performed two sets of experiments. In the first set of experiments, we consider a scenario that often arises in practice where a solution must be found within some given amount of computational resources. In these experiments we used a limit on the amount of CPU time. In such a scenario, the appropriate performance measure is the number of problems in the ensemble which are not solved. In the second set of experiments, we consider a scenario where a backtracking search is run to completion. In such a scenario, the appropriate performance measure is the expected time to solve all of the problems in the ensemble. We begin by presenting the experimental setup that is in common to the two sets of experiments, followed by the results of the experiments themselves.

5.1 Experimental Setup

We used instruction scheduling problems for multiple-issue pipelined processors in our experiments. Instruction scheduling is one of the most important steps in improving the performance of object code produced by a compiler. The task is to find a minimal length schedule for a basic block—a straight-line sequence of code with a single entry point and a single exit point—subject to precedence, latency, and resource constraints. We formulated a constraint programming model and

solved the instances using backtracking search. In the model, there is a variable for each instruction, the domains of the variables are the time cycles in which the instruction could be scheduled, and the constraints consist of linear inequalities, global cardinality constraints, and other specialized constraints. The scheduler was able to solve almost all of the basic blocks that we found in practice, including basic blocks with up to 2600 instructions. We refer the reader to [16] for further details on the problem, the model, and the backtracking algorithm. The point we wish to emphasize here is that, prior to our examination of restart strategies, considerable effort went into improving the constraint programming model, backtracking algorithm, and heuristics to reduce the number of unsolved problems.

To randomize the backtracking algorithm, the dynamic variable ordering heuristic randomly picked a variable from the top 5 variables (or fewer, if there were fewer variables left). The backtracking algorithm is capable of performing three levels of constraint propagation:

Level = 0	bounds consistency
Level = 1	singleton bounds consistency
Level = 2	singleton bounds consistency to a depth of two

We repeated the experiments for each level of constraint propagation. For our experiments, we used scheduling instances that arise from the SPEC 2000 and MediaBench benchmark suites, two standard real-world benchmarks in compiler research. These benchmark suites consist of source code for software packages that are chosen to be representative of a variety of programming languages and types of applications. We had available to us a total of 6,377 hard scheduling instances from 28 different software packages. For our sample of instances, or training set, we used all 927 of the hard scheduling instances from the galgel, gap, mpeg, and jpeg software packages. We chose these four software packages for two reasons. First, the instances give approximately fifteen percent of the scheduling instances and, second, these software packages provide a good cross section of the data as they include both integer and floating point instructions as well as a variety of programming languages and types of applications. For our test set, we used the remaining 5,450 hard scheduling instances.

For each instance in the training and test sets, we collected 1000 samples of its runtime distribution by each time running the randomized backtracking algorithm on the instance and recording the amount of time taken in seconds. The samples are censored in that we ran the backtracking algorithm with a timeout mechanism; if the instance was not solved within 10 minutes, the backtracking algorithm was terminated and the maximum amount of 10 minutes was recorded. (All times were recorded to 1/100 of a second, the resolution of the system clock.) These empirical runtime distributions were then used to learn and test the various restart strategies.

All of the runtime experiments were performed on a cluster which consists of 768 machines running Linux, each with 4 GB of RAM and four 2.2 GHz processors.

Table 2. Expected number of scheduling instances in the test set *not* solved within a deadline of 10 minutes, for various fixed cutoff strategies (see text); the Luby and Walsh strategies with default, estimated, and optimal parameter settings; and various levels of constraint propagation.

	Fixed cutoff			Luby			Walsh		
Level	t_k^*	$\hat{t}_{single}$	$t = 10$m	def.	est.	opt.	def.	est.	opt.
0	118.6	233.0	193.2	124.1	125.2	123.0	141.2	140.1	140.0
1	25.7	637.0	33.9	37.6	28.4	28.4	34.5	27.9	27.8
2	84.3	2,056.0	85.0	187.0	85.0	85.0	166.1	85.0	85.0

Table 3. Percentage increase in the expected number of scheduling instances in the test set *not* solved relative to the gold standard t_k^*, for various restart strategies and levels of constraint propagation.

	Fixed cutoff			Luby			Walsh		
Level	t_k^*	$\hat{t}_{single}$	$t = 10$m	def.	est.	opt.	def.	est.	opt.
0	0.0%	96.5%	62.9%	4.6%	5.6%	3.7%	19.0%	18.1%	18.0%
1	0.0%	2,378.6%	31.9%	46.3%	10.5%	10.5%	34.2%	8.6%	8.2%
2	0.0%	2,338.9%	0.8%	121.8%	0.8%	0.8%	97.0%	0.8%	0.8%
Max	0.0%	2,378.6%	62.9%	121.8%	10.5%	10.5%	97.0%	18.1%	18.0%

5.2 Experiment 1

In our first set of experiments, we set a time limit of 10 minutes per instance and determined whether learning good parameter settings from a training set can reduce the number of problems in the test set which are not solved.

To learn good parameter settings for the Luby and Walsh strategies, we discretized the scale s into orders of magnitude, $10^{-1}, \ldots, 10^5$, and the geometric r into $2, 3, \ldots, 10$ (Luby) and $1.1, 1.2, \ldots, 2$ (Walsh). The best parameter settings were then estimated by choosing the values that minimized the expected number of instances not solved for the training set. The strategies with the estimated parameter settings were then evaluated on the test set. Table 2 summarizes the results. Table 3 presents the same information except now stated in terms of percentage change. For comparison purposes, we show the results for three fixed cutoff strategies: (i) the gold standard strategy t_k^* where, for each instance k in the *test* set, the optimal fixed cutoff strategy for that instance is used; (ii) the fixed cutoff strategy $\hat{t}_{single}$ that would be learned from the training set by the method of Gagliolo and Schmidhuber [15]; and (iii) the fixed cutoff strategy where the cutoff is set equal to the deadline of 10 minutes; i.e., there are no restarts.

It can be seen that on this testbed the fixed cutoff strategy $\hat{t}_{single}$ performs poorly. Parameterizing the universal strategies and estimating good parameter settings can give quite reasonable performance improvements over the default or unparameterized universal strategies in some cases. Further, the parameter

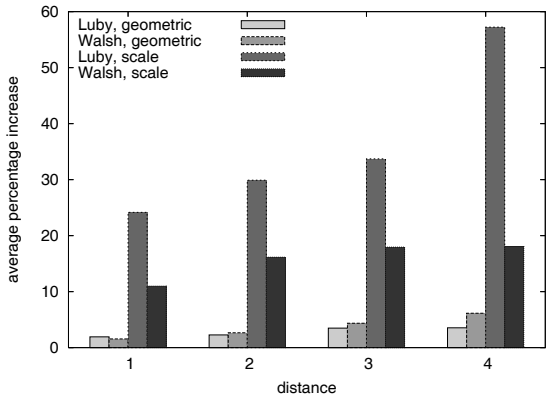

Fig. 1. Average (over all levels of constraint propagation) percentage increase in the expected number of instances *not* solved in the test set relative to the optimal parameter settings, for various distances.

settings estimated from the training set lead to performance that approaches that of the optimal parameter settings (the parameter settings that, among all possible parameter settings, lead to the best performance on the test set). As well, it is worth noting that in some cases the default parameter settings lead to worse performance than when there are no restarts.

We also measured the robustness of our approach for learning good universal restart strategies by performing a sensitivity analysis. To be broadly useful, our approach should not rely on extremely accurate estimates of the optimal settings of the parameters. Rather, there should be a wide range of values for the parameters that are effective and small changes in the accuracy of the estimates should give small changes in the overall performance of the strategy.

To measure the robustness, we began with the optimal parameter settings and systematically introduced inaccuracies in the parameters and observed the effect on performance. For each parameter setting that was a given distance from the optimal setting, we determined the percentage increase in the number of problems that could not be solved. For example, for a distance of 1, scale parameters that were one order of magnitude smaller and larger than the optimal setting were tested. The results are summarized in Figure 1. It can be seen that on this testbed: (i) the setting of the scale parameter is the most important for both the Luby and the Walsh strategies, (ii) estimates of the scale parameter that are off by one or two orders of magnitude are still effective, and (iii) the Walsh strategy is somewhat more robust than the Luby.

5.3 Experiment 2

In our second set of experiments, we removed the time limit and determined whether learning good parameter settings from a training set can reduce the time

Table 4. Expected time (days:hours:minutes) to solve *all* of the scheduling instances in the test set, for various fixed cutoff strategies (see text); the Luby and Walsh strategies with default, estimated, and optimal parameter settings; and various levels of constraint propagation.

	Fixed cutoff			Luby			Walsh		
Level	t_k^*	$\hat{t}_{single}$	$t = \infty$	def.	est.	opt.	def.	est.	opt.
0	4:13:54	∞	295:07:03	62:13:07	7:14:29	7:14:29	17:17:19	7:03:53	7:03:53
1	1:00:22	∞	53:11:59	13:23:17	1:08:12	1:08:12	3:16:35	1:06:17	1:06:17
2	3:11:46	∞	122:23:52	49:18:45	3:23:00	3:23:00	13:22:30	3:17:04	3:17:04

Table 5. Percentage increase in the expected time to solve *all* of the scheduling instances in the test set relative to the gold standard t_k^*, for various restart strategies and levels of constraint propagation.

	Fixed cutoff			Luby			Walsh		
Level	t_k^*	$\hat{t}_{single}$	$t = \infty$	def.	est.	opt.	def.	est.	opt.
0	0.0%	∞	6,348.4%	1265.8%	66.1%	66.1%	287.0%	56.4%	56.4%
1	0.0%	∞	5,169.1%	1275.9%	32.2%	32.2%	263.6%	24.3%	24.3%
2	0.0%	∞	3,423.8%	1326.3%	13.4%	13.4%	299.3%	6.3%	6.3%
Max	0.0%	∞	6,348.4%	1326.3%	66.1%	66.1%	299.3%	56.4%	56.4%

needed to solve all of the problems in the test set. Recall that in the runtime distributions that we gathered, the samples were censored in that the backtracking algorithm was terminated if an instance was not solved within 10 minutes. In this experiment we wished to run the backtracking algorithm to completion. It proved infeasible to actually do this on our benchmark instances—the instances that we let run without a timeout ran for days without terminating—and so we took a compromise approach as follows.

Gomes et al. [4] show that Pareto distributions of the form $F(t) = 1 - Ct^{-\alpha}$ are a good fit to runtime distributions that arise from backtracking algorithms with randomized heuristics when applied to a diverse set of problems. In our experimental runtime data, we replaced the timeouts by values sampled from the tail of a Pareto distribution with $C = 1$ and $\alpha = 0.5$. With these choices for C and α, of the instances that previously had timed out, approximately 59.2% of the instances are now "solved" within one hour, 91.9% are solved within one day, 98.9% are solved within one month, and all are solved within one year.

To learn good parameter settings for the Luby and Walsh strategies, we discretized the scale s into orders of magnitude, $10^{-1}, \ldots, 10^{7}$, and the geometric r into 2, 3, ..., 10 (Luby) and 1.1, 1.2, ..., 2 (Walsh). The best parameter settings were then estimated by choosing the values that minimized the time to solve all of the instances in the training set. The strategies with the estimated parameter settings were then evaluated on the test set. Table 4 summarizes the results. Table 5 presents the same information except now stated in terms of percentage change.

It can be seen that on this testbed parameterizing the universal strategies and estimating good parameter settings gives performance improvements over the default universal strategies in all cases (ranging from a minimum reduction of from 3 days down to 1 day to a maximum reduction of from 62 days down to 7 days). Further, the estimated parameter settings are accurate—in each case the estimated parameter settings turned out to be the optimal parameter settings. As well, we note that with just the default parameters, the Walsh strategy is much better than the Luby strategy. However, once we estimate and use good parameter settings, the Luby and Walsh strategies achieve quite comparable performance.

5.4 Discussion

We conclude this section with a discussion relating our experimental results with previous theoretical results, as at first glance it may appear that there is a conflict. Recall that Luby, Sinclair, and Zuckerman [1] show that, for any runtime distribution, the Luby strategy with default parameter settings is within a log factor of the optimal fixed cutoff strategy for that distribution. This may appear to contradict the experimental results that we report for the default Luby strategy. However, the theoretical result hides constant factors, which can be important in practice. The constant factors can in fact be large as one moves from the theoretician's time step to a more practical proxy such as clock time or number of backtracks. As well, we note that the optimality result does not hold in the case where limits are placed on computational resources, such as a deadline [17].

Recall also that Luby, Sinclair, and Zuckerman [1] show that, given no knowledge of the runtime distribution, no universal strategy can have a better bound than being within a log factor of optimal. This may appear to contradict the experimental results that we report for the universal strategies with estimated parameter settings. However, we note that the result does not hold in the case where we are allowed to sample from the ensemble—as we do here—and learn about the runtime distribution. As well, the theoretical result is a worst-case analysis and the proof relies on a pathological distribution which may not arise in practice.

6 Conclusions

We presented a theoretical worst-case analysis of the Walsh universal strategy. The analysis provides some insights into suitable ranges in practice for the geometric parameter of the strategy. We also presented an approach for learning good universal restart strategies in the commonly occurring scenario where an ensemble of instances is to be solved. We demonstrated the effectiveness and robustness of our approach through an extensive empirical evaluation on a real-world testbed. Together these results increase our theoretical understanding of universal restart strategies and increase their applicability in practice.

Acknowledgments. This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET).

References

1. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of Las Vegas algorithms. Information Processing Letters **47** (1993) 173–180
2. Harvey, W.D.: Nonsystematic backtracking search. PhD thesis, Stanford University (1995)
3. Gomes, C., Selman, B., Crato, N.: Heavy-tailed distributions in combinatorial search. In: Proceedings of the Third International Conference on Principles and Practice of Constraint Programming, Linz, Austria (1997) 121–135
4. Gomes, C., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. J. of Automated Reasoning **24** (2000) 67–100
5. Huang, J.: The effect of restarts on the efficiency of clause learning. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India (2007) 2318–2323
6. Walsh, T.: Search in a small world. In: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Stockholm (1999) 1172–1177
7. Zhan, Y.: Randomisation and restarts. MSc thesis, University of York (2001)
8. Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., Selman, B.: Dynamic restart policies. In: Proceedings of the Eighteenth National Conference on Artificial Intelligence, Edmonton (2002) 674–681
9. Ruan, Y., Horvitz, E., Kautz, H.: Restart policies with dependence among runs: A dynamic programming approach. In: Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming, Ithaca, New York (2002) 573–586
10. Ruan, Y., Horvitz, E., Kautz, H.: Hardness aware restart policies. In: IJCAI Workshop on Stochastic Search Algorithms (2003)
11. Bayardo Jr., R.J., Schrag, R.C.: Using CSP look-back techniques to solve real-world SAT instances. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence, Providence, Rhode Island (1997) 203–208
12. Wu, H., van Beek, P.: Restart strategies: Analysis and simulation (Doctoral Abstract). In: Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming, Kinsale, Ireland (2003) 1001
13. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Theory and Applications of Satisfiability (Selected and Revised Papers of SAT 2003), Lecture Notes in Computer Science 2919. (2004) 502–518
14. Ó Nualláin, B., de Rijke, M., van Benthem, J.: Ensemble-based prediction of SAT search behaviour. Electronic Notes in Discrete Mathematics **9** (2001)
15. Gagliolo, M., Schmidhuber, J.: Learning restarts strategies. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India (2007) 792–797
16. Malik, A.M., McInnes, J., van Beek, P.: Optimal basic block instruction scheduling for multiple-issue processors using constraint programming. In: Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, Washington, DC (2006) 279–287
17. van Moorsel, A.P.A., Wolter, K.: Analysis of restart mechanisms in software systems. IEEE Trans. on Software Engineering **32** (2006) 547–558