Restart Schedules for Ensembles of Problem Instances

Matthew Streeter Daniel Golovin Stephen F. Smith
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
{matts,dgolovin,sfs}@cs.cmu.edu

Abstract

The mean running time of a Las Vegas algorithm can often be dramatically reduced by periodically restarting it with a fresh random seed. The optimal restart schedule depends on the Las Vegas algorithm's run length distribution, which in general is not known in advance and may differ across problem instances. We consider the problem of selecting a single restart schedule to use in solving each instance in a set of instances. We present offline algorithms for computing an (approximately) optimal restart schedule given knowledge of each instance's run length distribution, generalization bounds for learning a restart schedule from training data, and online algorithms for selecting a restart schedule adaptively as new problem instances are encountered.

Introduction

Previous work has shown that the mean running time of a Las Vegas algorithm can often be reduced by periodically restarting the algorithm according to some schedule. Indeed, solvers based on chronological backtracking often exhibit heavy-tailed run length distributions, and restarts can yield order-of-magnitude improvements in performance (Gomes, Selman, & Kautz 1998).

In this paper we consider the following online problem. We are given a Las Vegas algorithm $\mathcal{A}$, which takes as input an instance x of some decision problem and returns a (provably correct) "yes" or "no" answer, but whose running time $T(x)$ is a random variable. We are then fed, one at a time, a sequence $\langle x_1, x_2, \ldots, x_n \rangle$ of problem instances to solve (note that the distribution of $T(x_i)$ may be different for different instances x_i). We solve each instance by running $\mathcal{A}$ and periodically restarting it according to some schedule, stopping as soon as we obtain an answer. A *restart schedule* is an infinite sequence $S = \langle t_1, t_2, \ldots \rangle$ of positive integers, whose meaning is "run $\mathcal{A}$ for t_1 time units; if this does not yield a solution then restart $\mathcal{A}$ and run for t_2 time units, ...". Our goal is to (adaptively) select restart schedules so as to minimize the total time required to solve all n instances.

In the special case when $n = 1$ (and no prior information about $\mathcal{A}$ or the problem instance x_1 is available), a near-optimal solution to this problem was given in a paper by Luby *et al.* (1993). Specifically, their paper gave a universal restart schedule that runs in expected time $O(\ell \log \ell)$, where ℓ is the expected time required by an optimal schedule for the pair $(\mathcal{A}, x_1)$. They further showed that any restart schedule requires expected time $\Omega(\ell \log \ell)$ for some pair $(\mathcal{A}, x_1)$, so that this guarantee is optimal to within constant factors.

For the more realistic case $n > 1$, however, much less is available in the way of theoretical results. Previous work either assumed that the Las Vegas algorithm's run length distribution is the same on each problem instance, or that it is one of k known distributions for some small k. In this paper we consider the problem of selecting restart schedules in full generality, where the Las Vegas algorithm may exhibit a different run length distribution on each instance. We address this problem in three settings: offline, learning-theoretic, and online, and in each setting we provide new theoretical guarantees. We demonstrate the power of our techniques experimentally on a logistics planning domain.

Motivations

To appreciate the relevance of this online problem, consider the two run length distributions depicted in Figure 1. Figure 1 shows the run length distribution of the SAT solver satz-rand on two Boolean formulae created by running a state-of-the-art planning algorithm, SATPLAN, on a logistics planning benchmark. To find a provably minimum-length plan, SATPLAN creates a sequence of Boolean formulae $\sigma_1, \sigma_2, \ldots$, where σ_i is satisfiable if and only if there exists a feasible plan of length $\leq i$. In this case the minimum plan length is 14. When run on the (satisfiable) formula σ_{14}, satz-rand exhibits a heavy-tailed run length distribution. There is about a 20% chance of solving the problem after running for 2 seconds, but also a 20% chance that a run will not terminate after having run for 1000 seconds. Restarting the solver every 2 seconds reduces the mean run length by more than an order of magnitude. In contrast, when satz-rand is run on the (unsatisfiable) instance σ_{13}, over 99% of the runs take at least 19 seconds, so the same restart policy would be ineffective. Restarts are still beneficial on this instance, however; restarting every 45 seconds reduces the mean run length by at least a factor of 1.5. Of course, when using a randomized SAT solver to solve a given formula for the first time one does not know its run length distribution on that formula, and thus one must select a restart schedule based on experience with previously-solved formulae.

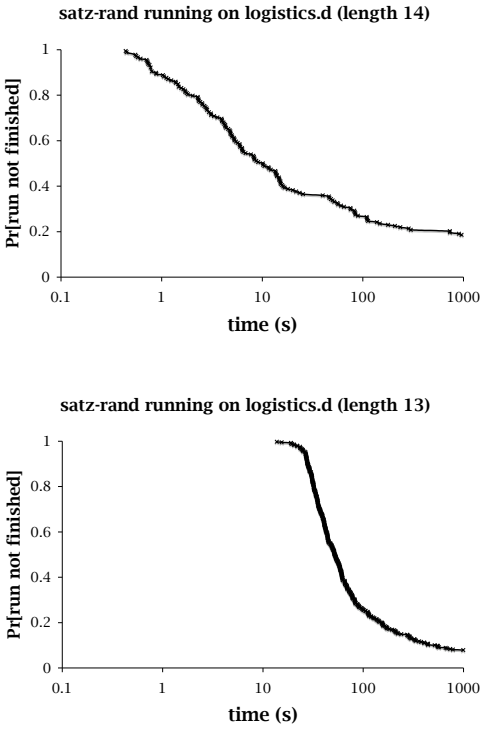

Figure 1: Run length distribution of satz-rand on two formulae created by SATPLAN in solving the logistics planning instance logistics.d (Gomes, Selman, & Kautz 1998). Each curve was estimated using 150 independent runs, and run lengths were capped at 1000 seconds.

Related Work

In the early 1990s, at least two papers studied the problem of selecting a restart schedule to use in solving a *single* problem instance, given no prior knowledge of the algorithm's run length distribution (this setting corresponds to the case $n = 1$ in our framework). The main results are summarized in the following theorem proved by Luby *et al.* (1993). Here $T_S(x)$ is a random variable equal to the time required to solve x when running $\mathcal{A}$ according to restart schedule S.

Theorem 1 (Luby *et al.* , 1993).

1. *For any instance x, the schedule S that minimizes $\mathbb{E}[T_S(x)]$ is a uniform restart schedule of the form $\langle t^*, t^*, \ldots, t^* \rangle$.*
2. *Let $\ell = \min_S \mathbb{E}[T_S(x)]$. The universal restart schedule*

$$S_{univ} = \langle 1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots \rangle$$

   has $\mathbb{E}[T_{S_{univ}}(x)] = O(\ell \log \ell)$.
3. *For any schedule S and any $\ell > 0$, it is possible to define a distribution of $T(x)$ such that $\mathbb{E}[T_S(x)] \geq \frac{1}{8} \ell \log_2 \ell$ (i.e., the worst-case performance of S_{univ} is optimal to within constant factors).*

Alt *et al.* (1996) gave related results, with a focus on minimizing tail probabilities rather than expected running time.

In the late 1990s, there was a renewed interest in restart schedules due to a paper by Gomes *et al.* (1998), which demonstrated that (then) state-of-the-art solvers for Boolean satisfiability and constraint satisfaction could be dramatically improved by randomizing the solver's decision-making heuristics and running the randomized solver with an appropriate restart schedule. In one experiment, their paper took a deterministic SAT solver called satz and created a version called satz-rand with a randomized heuristic for selecting which variable to branch on at each node in the search tree. They found that, on certain problem instances, satz-rand exhibited a heavy-tailed run length distribution. By periodically restarting it they obtained order-of-magnitude improvements in running time over both satz-rand (without restarts) and satz, the original deterministic solver. Their paper also demonstrated the benefit of randomization and restart for a state-of-the-art constraint solver.

One limitation of Theorem 1 is that it is "all or nothing": it either assumes complete knowledge of the run length distribution (in which case a uniform restart schedule is optimal) or no knowledge at all (in which case the universal schedule is optimal to within constant factors). Several papers have considered the case in which partial but not complete knowledge of the run length distribution is available. Ruan *et al.* (2002) consider the case in which each run length distribution is one of k known distributions, and give a dynamic programming algorithm for computing an optimal restart schedule. The running time of their algorithm is exponential in k, and thus it is practical only when k is small (in the paper the algorithm is described for $k = 2$). Kautz *et al.* (2002) considered the case in which, after running for some fixed amount of time, one observes a feature that gives the distribution of that run's length.

A recent paper by Gagliolo & Schmidhuber (2007) considered the problem of selecting restart schedules online in order to solve a sequence of problem instances as quickly as possible. Their paper treats the schedule selection problem as a 2-armed bandit problem, where one of the arms runs Luby's universal schedule and the other arm runs a schedule designed to exploit the empirical run length distribution of the instances encountered so far. Their strategy was designed to work well in the case where each instance has a similar run length distribution.

Although the focus of this paper is on restarting a single Las Vegas algorithm $\mathcal{A}$, our results extend naturally to the case when we have access to a set $\{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k\}$ of Las Vegas algorithms and may choose among them each time we restart. In a companion paper (Streeter, Golovin, & Smith 2007b) we consider the case of multiple heuristics in more detail.

Preliminaries

As discussed in the introduction, we consider running a single Las Vegas algorithm $\mathcal{A}$, which takes as input a problem instance x and a random seed $s \in \{0, 1\}^\infty$. When run on a particular problem instance, $\mathcal{A}$ always returns a (provably correct) "yes" or "no" answer, but the time it requires depends on the random seed. The random variable $T(x)$ equals the time that $\mathcal{A}$ runs for before returning an answer. We assume $T(x)$ is always a positive integer.

A *restart schedule* is an infinite sequence of positive integers $\langle t_1, t_2, t_3, \ldots \rangle$ whose meaning is "run $\mathcal{A}$ for t_1 time units; if it does not terminate then restart and run for t_2 time units, ...". We denote by $\mathcal{S}_{rs}$ the set of all restart schedules. For any schedule $S \in \mathcal{S}_{rs}$, the random variable $T_S(x)$ equals the time that S runs for before terminating.

We assume a bound B on the time a schedule is allowed to run. The cost associated with running schedule S on instance x_i is $c_i(S) \equiv \mathbb{E}[\min\{B, T_S(x_i)\}]$.

Profiles

The *profile* $\mathcal{P}_S(t)$ lists, in descending order, the lengths of the runs that restart schedule S has performed after it has been run for t time units (without finding a solution). For example, if S is the geometric restart schedule $\langle 1, 2, 4, 8, 16, \ldots \rangle$ then $\mathcal{P}_S(3) = \langle 2, 1 \rangle$ and $\mathcal{P}_S(6) = \langle 3, 2, 1 \rangle$ (where at time 6 the current run of length 4 is only $\frac{3}{4}$ completed).

Overloading notation, for any profile $\mathcal{P} = \langle \tau_1, \tau_2, \ldots, \tau_k \rangle$ let $c_i(\mathcal{P}) = \prod_{j=1}^{k} \mathbb{P}[T(x_i) > \tau_j]$. Thus $c_i(\mathcal{P}_S(t))$ equals the probability that S fails to find a solution after running for t time steps. By linearity of expectation,

$$c_i(S) = \sum_{t=0}^{B-1} c_i(\mathcal{P}_S(t)) \,. \tag{1}$$

We refer to the quantity $\sum_{j=1}^{k} \tau_j$ as the *size* of $\mathcal{P}$.

Shortest path formulation

For any set $\mathcal{S}$ of schedules, define the *profile span* of $\mathcal{S}$ as $\mathcal{P}^+(\mathcal{S}) = \{\mathcal{P}_S(t) : S \in \mathcal{S}, 0 \le t < B\}$. The *profile graph* $G(\mathcal{S})$ is defined as follows. The vertex set is $\mathcal{P}^+(\mathcal{S}) \cup \{v^*\}$. The edges are defined by the following rule: for any restart schedule $S \in \mathcal{S}$, the vertices $\langle \mathcal{P}_S(0), \mathcal{P}_S(1), \mathcal{P}_S(2), \ldots \mathcal{P}_S(B-1), v^* \rangle$ form a directed path in the graph. The weight assigned to the directed edge $(\mathcal{P}, v)$ is $\sum_{i=1}^{n} c_i(\mathcal{P})$. Using (1), we have that for any schedule S, the weight assigned to the corresponding path is $\sum_{i=1}^{n} c_i(S)$. This yields the following lemma.

Lemma 1. *For any set $\mathcal{S}$ of restart schedules, the schedule $S^* = \arg\min_{S \in \mathcal{S}} \sum_{i=1}^{n} c_i(S)$ may be found by computing a shortest path from v^0 to v^* in $G(\mathcal{S})$, where $v^0 = \langle\rangle$ is the empty profile.*

α-Regularity

For any $\alpha > 1$, let $Z_\alpha = \{\lfloor \alpha^i \rfloor : i \in \mathbb{Z}^+\}$ be the set of integers that are powers of α rounded down to the nearest integer. We say that a profile $\langle \tau_1, \tau_2, \ldots, \tau_k \rangle$ is α-regular if all τ_j are in Z_α and if, for each τ_j, the number of runs of length τ_j is also a member of Z_α. For example, the profiles $\langle 4, 1, 1 \rangle$ and $\langle 4, 2, 1, 1, 1, 1 \rangle$ are 2-regular but the profiles $\langle 3, 1 \rangle$ and $\langle 2, 1, 1, 1 \rangle$ are not.

We say that a restart schedule $S = \langle t_1, t_2, t_3, \ldots \rangle$ is α-regular if $t_j \in Z_\alpha$ for all j and if, whenever $t_j \neq t_{j+1}$, the number of runs of length t_j performed so far is in Z_α (i.e., $|\{j' \le j : t_{j'} = t_j\}| \in Z_\alpha$). For example, the geometric restart schedule $\langle 1, 2, 4, 8, 16, \ldots \rangle$ is 2-regular, as is the following slightly modified version of Luby's universal schedule: $\langle 1, 1, 2, 1, 1, 2, 4, 1, 1, 1, 2, 2, 4, 8, \ldots \rangle$; however, the schedule $\langle 1, 2, 1, 2, 1, 2, \ldots \rangle$ is not. Equivalently, S is α-regular if, whenever $t_j \neq t_{j+1}$, the profile $\mathcal{P}_S(\sum_{j'=1}^{j} t_{j'})$ is α-regular. We denote by $\mathcal{S}_{rs}^{\alpha}$ the set of all α-regular schedules.

The following lemma (proved in Appendix A) shows that an arbitrary schedule can be "rounded up" to an α-regular schedule while introducing at most a factor α^2 overhead.

Lemma 2. *For any schedule S and $\alpha > 1$, there exists an α-regular schedule S_α such that, for any instance x, $\mathbb{E}[T_S(x)] \le \alpha^2 \mathbb{E}[T_{S_\alpha}(x)]$.*

Offline Algorithms

In the offline setting we are given as input the values of $p_i(t) \equiv \mathbb{P}[T(x_i) \le t]$ for all i ($1 \le i \le n$) and t ($1 \le t \le B$), and we wish to compute the restart schedule

$$S^* = \arg\min_{S \in \mathcal{S}_{rs}} \sum_{i=1}^{n} c_i(S) \,.$$

Although we have not been able to determine whether this problem is NP-hard, we do believe that designing an algorithm to solve it is non-trivial. One simple idea that does *not* work is to compute an optimal restart schedule for the averaged distribution $p(t) = \frac{1}{n} \sum_{i=1}^{n} p_i(t)$. To see the flaw in this idea, consider the following example with $n = 2$: $T(x_1) = 1$ with probability $\frac{1}{2}$ and is 1000 with probability $\frac{1}{2}$, while $T(x_2) = 1000$ with probability 1. The optimal schedule for the averaged distribution is the uniform schedule $\langle 1, 1, 1, \ldots \rangle$, however this schedule *never* solves x_2.

A quasi-polynomial time approximation scheme

Our first approximation algorithm simply combines Lemmas 1 and 2 in the natural way, namely it obtains an α^2-approximation to the optimal restart schedule by computing a shortest path in the profile graph $G(S_{rs}^{\alpha})$. To bound the time complexity, we show that the number of α-regular profiles with size $< B$ is $O((\log_\alpha B)^{\log_\alpha B}) = O(B^{\log_\alpha \log_\alpha B})$, then argue that the algorithm can be implemented using $O(n \log_\alpha B)$ time per α-regular profile (for a formal proof, see Appendix A).

Theorem 2. *There exists an algorithm that runs in time $O(n(\log_\alpha B) B^{\log_\alpha \log_\alpha B})$ and returns an α^2-approximation to the optimal restart schedule.*

A greedy approximation algorithm

Our second approximation algorithm runs in polynomial time and returns a 4-approximation to the optimal schedule. The algorithm proceeds in a series of epochs, each of which selects a restart threshold to add to the schedule. At the beginning of epoch z, r_i equals the probability that instance x_i remains unsolved after performing runs of lengths $t_1, t_2, \ldots, t_{z-1}$. The threshold t_z is selected so as to maximize the expected number of additional instances solved per unit time.

Greedy algorithm for constructing restart schedules:
1. Initialize $r_i = 1 \; \forall i, 1 \leq i \leq n$.
2. For z from 1 to B:
   (a) Find the threshold $t_z \in \{1, 2, \ldots, B\}$ that maximizes
   $$\frac{1}{t_z} \sum_{i=1}^{n} r_i p_i(t_z) \; .$$
   (b) Set $r_i = r_i \cdot (1 - p_i(t_z)) \; \forall i$.
3. Return the schedule $\langle t_1, t_2, \ldots t_B \rangle$.

The performance of the algorithm is summarized in the following theorem, which is proved in Appendix B (Streeter, Golovin, & Smith 2007a).

Theorem 3. *The greedy approximation algorithm returns a 4-approximation to the optimal restart schedule.*

As written, the greedy algorithm requires $O(B)$ time per epoch. If each threshold t_z is restricted to be a power of α, then each epoch requires time $O(\log_\alpha B)$ and we obtain a 4α-approximation (as the proof of Theorem 3 shows).

Learning Restart Schedules

To apply the offline algorithms of the previous section in practice, we might collect a set of problem instances to use as training data, compute an (approximately) optimal restart schedule for the training instances, and then use this schedule to solve additional test instances. Under the assumption that the training and test instances are drawn (independently) from a fixed probability distribution, we would then like to know how much training data is required so that (with high probability) our schedule performs nearly as well on test data as it did on the training data. In our setting two distinct questions arise: how many training instances do we need, and how many runs must we perform on each training instance in order to estimate its run length distribution to sufficient accuracy? We deal with each question separately in the following subsections.

Estimating profile costs

In this section our objective is the following: given an instance x_i, we would like to spend as little time as possible running $\mathcal{A}$ on x_i in order to obtain a function $\bar{c}_i : \mathcal{P}^+(\mathcal{S}_{rs}) \to [0, 1]$ with the following property: for any profile $\mathcal{P} \in \mathcal{P}^+(\mathcal{S}_{rs})$, $\mathbb{E}[\bar{c}_i(\mathcal{P})] = c_i(\mathcal{P})$ (i.e., $\bar{c}_i(\mathcal{P})$ is an unbiased estimate of $c_i(\mathcal{P})$).

To obtain such a function, we will perform B independent runs of $\mathcal{A}$ on x_i, where the j^{th} run is performed with a time limit of $\frac{B}{j}$ (so the total running time is at most $\sum_{j=1}^{B} \frac{B}{j} = O(B \log B)$). Let T_j be the *actual* time required by the j^{th} run (whereas we only have knowledge of $\min\{T_j, \frac{B}{j}\}$), and call the tuple $\mathcal{T} = \langle T_1, T_2, \ldots, T_B \rangle$ a *trace*. For any profile $\mathcal{P} = \langle \tau_1, \tau_2, \ldots, \tau_k \rangle$, we say that $\mathcal{T}$ encloses $\mathcal{P}$ if $T_j > \tau_j$ for all j (see Figure 2). Our estimate is

$$\bar{c}_i(\mathcal{P}) = \begin{cases} 1 & \text{if } \mathcal{T} \text{ encloses } \mathcal{P} \\ 0 & \text{otherwise.} \end{cases}$$

The estimate is unbiased because $\mathbb{E}[\bar{c}_i(\mathcal{P})] = \prod_{j=1}^{k} \mathbb{P}[T_j > \tau_j] = c_i(\mathcal{P})$. Furthermore, the estimate can be computed given only knowledge of $\min\{T_j, \frac{B}{j}\}$. This is true because if $\mathcal{P} = \langle \tau_1, \tau_2, \ldots, \tau_k \rangle$ is a profile with size $< B$, then for each j we have $\tau_j < \frac{B}{j}$ (recall that the sequence $\langle \tau_1, \tau_2, \ldots \tau_k \rangle$ is non-increasing by definition).

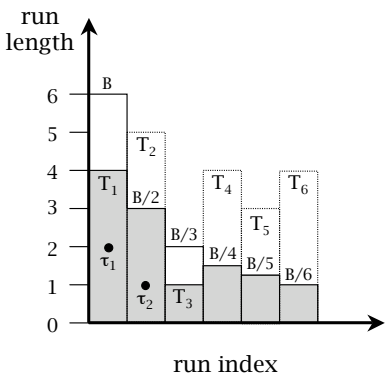

Figure 2: An illustration of our estimation procedure. The profile $\langle \tau_1, \tau_2 \rangle$ (dots) is enclosed by the trace $\langle T_1, T_2, T_3, T_4, T_5, T_6 \rangle$.

Although the estimate just described is unbiased, it is somewhat crude in that the estimate of $c_i(\mathcal{P})$ (which is a probability) is always 0 or 1. The following lemma (which is proved in Appendix A) gives a more refined unbiased estimate which we will use in our experimental evaluation. As in the previous paragraph, computing the estimate only requires knowledge of $\min\{T_j, \frac{B}{j}\}$ for each j.

Lemma 3. *For any profile $\mathcal{P} = \langle \tau_1, \tau_2, \ldots, \tau_k \rangle$ of size $< B$, define $L_j(\mathcal{P}) = \{j' : 1 \leq j' \leq B, \frac{B}{j'} > \tau_j\}$. Then the quantity*

$$\bar{c}_i(\mathcal{P}) = \prod_{j=1}^{k} \frac{|\{j' \in L_j(\mathcal{P}) : T_{j'} > \tau_j\}| - j + 1}{|L_j(\mathcal{P})| - j + 1}$$

is an unbiased estimate of $c_i(\mathcal{P})$ (i.e., $\mathbb{E}[\bar{c}_i(\mathcal{P})] = c_i(\mathcal{P})$).

Sample complexity bounds

Let $\{x_1, x_2, \ldots, x_m\}$ be a set of m training instances drawn independently from some distribution. Let $\mathcal{S}$ be an arbitrary set of schedules. For any schedule $S \in \mathcal{S}$, let $c(S)$ be the expected cost of S on a random instance drawn from the same distribution, and similarly for any profile $\mathcal{P}$ let $c(\mathcal{P})$ be its expected cost on a random instance.

Suppose we run $\mathcal{A}$ for $O(B \log B)$ time on each of the m training instances to obtain functions $\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_m$, as per the discussion in the previous section. Because the estimates are unbiased, we have $\mathbb{E}[\bar{c}_i(\mathcal{P})] = \mathbb{E}[c_i(\mathcal{P})] = c(\mathcal{P})$. For any profile $\mathcal{P}$ (of size $< B$), let $\bar{c}(\mathcal{P}) = \frac{1}{m} \sum_{i=1}^{m} \bar{c}_i(\mathcal{P})$. Then $\bar{c}(\mathcal{P})$ is the average of m independent identically distributed random variables, each of which has range $[0, 1]$ and expected value $c(\mathcal{P})$. Thus by Hoeffding's inequality,

$$\mathbb{P}\left[|\bar{c}(\mathcal{P}) - c(\mathcal{P})| \geq \epsilon\right] \leq 2 \exp\left(-2m\epsilon^2\right) \; .$$

It follows that for any $\delta' > 0$, $m = O(\frac{1}{\epsilon^2} \ln \frac{1}{\delta'})$ training instances are required to ensure $\mathbb{P}\left[|\bar{c}(\mathcal{P}) - c(\mathcal{P})| \geq \epsilon\right] < \delta'$.

For any schedule S, define $\bar{c}_i(S) = \sum_{t=0}^{B-1} \bar{c}_i(\mathcal{P}_S(t))$, in analogy to (1). Using (1), we have

$$\max_{S \in \mathcal{S}} |\bar{c}(S) - c(S)| \leq B \cdot \max_{\mathcal{P} \in \mathcal{P}^+(\mathcal{S})} |\bar{c}(\mathcal{P}) - c(\mathcal{P})| .$$

Setting $\delta' = \frac{\delta}{|\mathcal{P}^+(\mathcal{S})|}$ and using the union bound yields the following theorem.

Theorem 4. *If the number of training instances satisfies* $m \geq m_0(\epsilon, \delta, \mathcal{S}) = O\left(\frac{1}{\epsilon^2} \ln \frac{|\mathcal{P}^+(\mathcal{S})|}{\delta}\right)$*, the inequality*

$$\max_{S \in \mathcal{S}} |\bar{c}(S) - c(S)| \leq \epsilon B$$

holds with probability at least $1 - \delta$.

Using Theorem 4, we can obtain sample complexity bounds for sets $\mathcal{S}$ of particular interest by bounding $\ln |\mathcal{P}^+(\mathcal{S})|$. We first consider $\mathcal{S}_{rs}$. Let $p(m)$ denote the number of profiles of size exactly m, so $|\mathcal{P}^+(\mathcal{S}_{rs})| = \sum_{m=0}^{B-1} p(m)$. Then $p(m)$ is the famous *partition function* that counts the number of unordered partitions of m, and Hardy & Ramanujan (1918) proved that $\ln p(m) = \Theta(\sqrt{m})$. Thus $\ln |\mathcal{P}^+(\mathcal{S}_{rs})| = \Theta(\sqrt{B})$.

We next consider $\mathcal{S}^{\alpha}_{rs}$. Let N denote the number of schedules in $\mathcal{S}^{\alpha}_{rs}$ with distinct behavior during the first B time steps. It is not hard to see that $N = O((\log_\alpha B)^{2 \log_\alpha B})$, and it is trivially true that $|\mathcal{P}^+(\mathcal{S}^{\alpha}_{rs})| \leq BN$, so $\ln |\mathcal{P}^+(\mathcal{S}^{\alpha}_{rs})| = O(\log_\alpha B \log\log_\alpha B)$. Thus we have the following corollary.

Corollary 1. *Let the function* $m_0(\epsilon, \delta, \mathcal{S})$ *be defined as in Theorem 4. Then* $m_0(\epsilon, \delta, \mathcal{S}_{rs}) = O\left(\frac{1}{\epsilon^2}\left(\sqrt{B} + \ln \frac{1}{\delta}\right)\right)$ *and* $m_0(\epsilon, \delta, \mathcal{S}^{\alpha}_{rs}) = O\left(\frac{1}{\epsilon^2}\left(\log_\alpha B(\log\log_\alpha B) + \ln \frac{1}{\delta}\right)\right)$.

Corollary 1 and Lemma 2 suggest the following procedure: compute an (approximately) optimal restart schedule for the m training instances (using any algorithm or heuristic whatsoever) then round the schedule up to an α-regular schedule (introducing at most a factor α^2 overhead), where α is chosen so that Corollary 1 applies for some desired ϵ and δ. The rounding step prevents overfitting, and its overhead is fairly low.

Online Strategies

One limitation of Theorem 4 is that it requires us to draw training (and test) instances independently at random from a fixed probability distribution. In practice, the distribution might change over time and successive instances might not be independent. To illustrate this point, consider again the example of using SATPLAN to solve one or more planning problems. To solve a particular planning problem, SATPLAN generates a sequence $\langle \sigma_1, \sigma_2, \ldots \rangle$ of Boolean formulae that are not at all independent. To optimize SATPLAN's performance, we would like to learn a restart schedule for the underlying SAT solver on-the-fly, without making strong assumptions about the sequence of formulae that are fed to it.

In this section we consider the problem of selecting restart schedules in a worst-case online setting. In this setting we are given as input a set $\mathcal{S}$ of restart schedules. We are then fed, one at a time, a sequence $\langle x_1, x_2, \ldots, x_n \rangle$ of problem instances to solve. Prior to receiving instance x_i, we must select a restart schedule $S_i \in \mathcal{S}$. We then use S_i to solve x_i and incur cost C_i equal to the time we spend running $\mathcal{A}$ on x_i. Our *regret* at the end of n rounds is equal to

$$\frac{1}{n}\left(\mathbb{E}\left[\sum_{i=1}^{n} C_i\right] - \min_{S \in \mathcal{S}} \sum_{i=1}^{n} c_i(S)\right) \tag{2}$$

where the expectation is over two sources of randomness: the random bits supplied to $\mathcal{A}$, but also any random bits used by our schedule-selection strategy. That is, regret is $\frac{1}{n}$ times the difference between the expected total cost incurred by our strategy and that of the optimal schedule for the (unknown) set of n instances. A strategy's worst-case regret is the maximum value of (2) over all instance sequences of length n (i.e., over all sequences of n run length distributions). A *no-regret strategy* has worst-case regret that is $o(1)$ as a function of n.

Assume for the moment that $|\mathcal{S}|$ is small enough that we would not mind using $O(|\mathcal{S}|)$ time or space for decision-making. In this case one option is to treat our online problem as an instance of the "nonstochastic multiarmed bandit problem" and use the **Exp3** algorithm of Auer *et al.* (2002) to obtain regret $O\left(B\sqrt{\frac{1}{n}|\mathcal{S}|}\right) = o(1)$.

To obtain regret bounds with a better dependence on $|\mathcal{S}|$, we use a version of the "label-efficient forecaster" of Cesa-Bianchi *et al.* (2005). Applied to our online problem, this strategy behaves as follows. Given an instance x_i, with probability γ the strategy *explores* by computing (an unbiased estimate of) $c_i(S)$ for each $S \in \mathcal{S}$. As per the discussion leading up to Lemma 3, we can accomplish each exploration step by running $\mathcal{A}$ for time at most $\sum_{j=1}^{B} \frac{B}{j} = O(B \log B)$, independent of $|\mathcal{S}|$. With probability $1 - \gamma$, the strategy *exploits* by selecting a schedule at random from a distribution in which schedule S is assigned probability proportional to $\exp(-\eta \bar{c}(S))$, where $\bar{c}(S)$ is an unbiased estimate of $\sum_{l=1}^{i-1} c_l(S)$ and η is a learning rate parameter. Adapting Theorem 1 of Cesa-Bianchi *et al.* (2005) to our setting yields the following regret bound, which depends only logarithmically on $|\mathcal{S}|$.

Theorem 5. *The label-efficient forecaster with learning rate* $\eta = \left(\frac{\ln |\mathcal{S}|}{n\sqrt{2H_B}}\right)^{2/3}$ *and exploration probability* $\gamma = \sqrt{\frac{\eta}{2H_B}}$ *has regret at most* $2B\left(\frac{2H_B \ln |\mathcal{S}|}{n}\right)^{1/3}$*, where* $H_B \equiv \sum_{j=1}^{B} \frac{1}{j} = O(\log B)$.

Handling large $|\mathcal{S}|$

A naïve implementation of the label-efficient forecaster requires $O(|\mathcal{S}|)$ time and space for decision-making on each instance. To reduce the decision-making time, we again exploit the shortest path formulation. Specifically, we can use the dynamic programming equations described by György *et*

al. (2006) for the *online shortest paths* problem. Using this technique, the total decision-making time (over n instances) is of the same order as the time required to solve the offline shortest path problem (e.g., for $S = \mathcal{S}_{rs}^{\alpha}$, the time complexity is given in Theorem 2).

Experimental Evaluation

Following Gagliolo & Schmidhuber (2007), we evaluate our techniques by using them to construct restart schedules for the SAT solver satz-rand. We note that satz-rand is at this point a relatively old SAT solver, however it has the following key feature: successive runs of satz-rand on the same problem instance are *independent*, as required by our theoretical results. More modern solvers (e.g., MiniSat) also make use of restarts but maintain a repository of conflict clauses that is shared among successive runs on the same instance, violating this independence assumption.

To generate a set of benchmark formulae, we use the *blackbox* instance generator to generate 80 random logistics planning problems, using the same parameters that were used to generate the instance logistics.d from the paper by Gomes *et al.* (1998). We then used SATPLAN to find an optimal plan for each instance, and saved the Boolean formulae it generated. This yielded a total of 242 Boolean formulae.[1] We then performed $B = 1000$ runs of satz-rand on each formula, where the j^{th} run was performed with a time limit of $\frac{B}{j}$ as per the discussion leading up to Theorem 4.

We evaluated (a) the schedule returned by the greedy approximation algorithm, (b) uniform schedules of the form $\langle t, t, t, \ldots \rangle$ for each $t \in \{1, 2, \ldots, B\}$; (c) geometric restart schedules of the form $\langle \beta^0, \beta^1, \beta^2, \ldots \rangle$ for each $\beta \in \{1.1^k : 1 \leq k \leq \lceil \log_{1.1} B \rceil\}$ and (d) Luby's universal restart schedule. We estimated the expected CPU time required by each schedule using the function $\bar{c}$ described in Lemma 3.

Table 1: Performance of various restart schedules (cross-validation results are parenthesized).

Restart schedule	**Avg. CPU (s)**
Greedy schedule	21.9 (22.8)
Best geometric	23.9
Best uniform	33.9
Luby's universal schedule	37.2
No restarts	74.1

Table 1 gives the average CPU time required by each of the schedules we evaluated. The schedule returned by the greedy approximation algorithm had the smallest mean running time. The greedy schedule was 1.7 times faster than Luby's universal schedule, 1.5 times faster than the best uniform schedule (which used threshold $t = 85$), and 1.1 times faster than the best geometric schedule (which set $\beta \approx 1.6$). The average CPU time for a schedule that performed no restarts was about 3.4 times that of the greedy schedule, but is likely to be much worse due to the fact that run lengths were capped at 1000 seconds.

Examining Table 1, one may be concerned that the greedy algorithm was run using the same estimated run length distributions that were later used to estimate its expected CPU time. To address the possibility of overfitting, we also evaluated the greedy algorithm using leave-one-out cross-validation.[2] The estimated mean CPU time increased by about 4% under leave-one-out cross-validation.

Conclusions

We addressed the problem of selecting a schedule for restarting a Las Vegas algorithm in order to minimize the time required to solve each instance in a set of instances. We considered the schedule selection problem in offline, learning-theoretic, and online settings, and derived new theoretical results in each setting. Experimentally, we used one of our offline algorithms to compute an improved restart schedule for running satz-rand on Boolean formulae derived from logistics planning problems.

Acknowledgment. This work was supported in part by NSF ITR grants CCR-0122581 and IIS-0121678 and by DARPA under Contract #FA8750-05-C-0033.

References

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

Auer, P.; Cesa-Bianchi, N.; Freund, Y.; and Schapire, R. E. 2002. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing* 32(1):48–77.

Cesa-Bianchi, N.; Lugosi, G.; and Stoltz, G. 2005. Minimizing regret with label efficient prediction. *IEEE Transactions on Information Theory* 51:2152–2162.

Gagliolo, M., and Schmidhuber, J. 2007. Learning restart strategies. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, 792–797.

Gomes, C.; Selman, B.; and Kautz, H. 1998. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 431–437.

György, A., and Ottucsák, G. 2006. Adaptive routing using expert advice. *The Computer Journal* 49(2):180–189.

Hardy, G. H., and Ramanujan, S. 1918. Asymptotic formulae in combinatory analysis. *Proceedings of the London Mathematical Society* 17:75–115.

Kautz, H.; Ruan, Y.; and Horvitz, E. 2002. Dynamic restarts. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 674–681.

[1] The number of generated formulae is less than the sum of the minimum plan lengths, because SATPLAN can trivially reject some plan lengths without invoking a SAT solver.

[2] Leave-one-out cross-validation is performed as follows: for each instance, we set that instance aside and run the greedy algorithm on the remaining data to obtain a schedule to use in solving that instance.

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

Ruan, Y.; Horvitz, E.; and Kautz, H. 2002. Restart policies with dependence among runs: A dynamic programming approach. In *Proceedings of the Eighth International Conference on Principles and Practice of Constraint Programming*, 573–586.

Streeter, M.; Golovin, D.; and Smith, S. F. 2007a. Appendix B: Proof of theorem 3. Available at `http://www.cs.cmu.edu/~matts/aaai07_appendixB.pdf`.

Streeter, M.; Golovin, D.; and Smith, S. F. 2007b. Combining multiple heuristics online. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*.

Appendix A: Proofs

Lemma 2. *For any schedule S and $\alpha > 1$, there exists an α-regular schedule S_α such that, for any instance x, $\mathbb{E}[T_S(x)] \leq \alpha^2 \mathbb{E}[T_{S_\alpha}(x)]$.*

Proof. For any profile $\mathcal{P} = \langle \tau_1, \tau_2, \ldots, \tau_k \rangle$, let $\lceil \mathcal{P} \rceil$ be the α-regular profile obtained by rounding each τ_j up to the nearest member of Z_α, then rounding the number of runs of each length up to the nearest member of Z_α. For example if $\alpha = 2$ and $\mathcal{P} = \langle 4, 3, 3, 2 \rangle$, then $\lceil \mathcal{P} \rceil = \langle 4, 4, 4, 4, 2 \rangle$. Note that the size of $\lceil \mathcal{P} \rceil$ is at most α^2 times the size of $\mathcal{P}$.

Consider the set $\{t : \lceil \mathcal{P}_S(t) \rceil \neq \lceil \mathcal{P}_S(t-1) \rceil\}$, and let $\langle t_1^*, t_2^*, t_3^* \ldots \rangle$ be a list of its elements in ascending order. Let S_α be the unique α-regular schedule that passes through the sequence of profiles

$$\lceil \mathcal{P}_S(t_1^*) \rceil, \lceil \mathcal{P}_S(t_2^*) \rceil, \lceil \mathcal{P}_S(t_3^*) \rceil, \ldots$$

We claim that for any time t,

$$\mathbb{P}[T_S(x) \leq t] \leq \mathbb{P}[T_{S_\alpha}(x) \leq \alpha^2 t]\,. \tag{3}$$

This follows from the fact that S_α passes through the profile $\lceil \mathcal{P}_S(t) \rceil$, which has size at most $\alpha^2 t$. Thus by time $\alpha^2 t$, S_α has done all the work that S has done at time t and more. The fact that (3) holds for all t implies $\mathbb{E}[T_S(x)] \leq \alpha^2 \mathbb{E}[T_{S_\alpha}(x)]$. □

Theorem 2. *There exists an algorithm that runs in time $O(n(\log_\alpha B) B^{\log_\alpha \log_\alpha B})$ and returns an α^2-approximation to the optimal restart schedule.*

Proof. Our algorithm obtains an α^2-approximation to the optimal restart schedule by finding (a truncated version of) the optimal schedule in S^α_{rs}, using a shortest path algorithm on a suitable graph $G = (V, E)$. The vertices of G consist of all α-regular profiles of size at most B, plus an additional vertex v^*. For each profile $\mathcal{P} \in V$, and each $r \in Z_\alpha \cap [1, B)$, we add an edge from $\mathcal{P}$ to the profile $\mathcal{P}'$ obtained from $\mathcal{P}$ by increasing the number of runs of length r in $\mathcal{P}$ to the next largest value in Z_α, assuming $\mathcal{P}' \in V$. For example, with $\alpha = 3$ there would be an edge $(\langle 3, 1 \rangle, \langle 3, 3, 3, 1 \rangle)$ corresponding to $r = 3$. If the profile $\mathcal{P}'$ derived this way has size greater than B, we add an edge $(\mathcal{P}, v^*)$ instead of $(\mathcal{P}, \mathcal{P}')$. Each edge $(\mathcal{P}, \mathcal{P}')$ is weighted to represent the expected cost incurred while executing the runs necessary to reach profile $\mathcal{P}'$ from $\mathcal{P}$. Each edge of the form $(\mathcal{P}, v^*)$ derived from some pair of profiles $(\mathcal{P}, \mathcal{P}')$ will be weighted to represent the expected cost incurred while executing runs necessary to reach profile $\mathcal{P}'$ from $\mathcal{P}$ if we stop execution after $B - \text{size}(\mathcal{P})$ steps. We then find the shortest path from the empty profile to v^*, and output the corresponding schedule.

If we precompute $\sum_{a=1}^{t} \mathbb{P}[T(x_i) > a]$, for all i and t, and precompute $\mathbb{P}[T(x_i) > t]^r$ for all i, t and r, then we can collectively evalute all the edge weights in $O(n)$ time per edge. The overall time complexity is dominated by the time to assign edge weights, which is $O(n|E|)$. Assuming for simplicity that B is a power of α, it is easy to see that the number of α-regular profiles of size at most B is at most $(\log_\alpha B)^{\log_\alpha B} = B^{\log_\alpha \log_\alpha B}$ and that each profile has at most $(\log_\alpha B)$ outgoing edges, which proves the theorem. □

Lemma 3. *For any profile $\mathcal{P} = \langle \tau_1, \tau_2, \ldots, \tau_k \rangle$ of size $< B$, define $L_j(\mathcal{P}) = \{j' : 1 \leq j' \leq B, \frac{B}{j'} > \tau_j\}$. Then the quantity*

$$\bar{c}_i(\mathcal{P}) = \prod_{j=1}^{k} \frac{|\{j' \in L_j(\mathcal{P}) : T_{j'} > \tau_j\}| - j + 1}{|L_j(\mathcal{P})| - j + 1}$$

is an unbiased estimate of $c_i(\mathcal{P})$ (i.e., $\mathbb{E}[\bar{c}_i(\mathcal{P})] = c_i(\mathcal{P})$).

Proof. Given the trace $\mathcal{T} = \langle T_1, T_2, \ldots, T_B \rangle$, suppose we construct a new trace $\mathcal{T}'$ by randomly permuting the elements of $\mathcal{T}$ using the following procedure (the procedure is well-defined assuming $|L_j(\mathcal{P})| \geq j$ for each j, which follows from the fact that $\tau_j < \frac{B}{j}$ if $\mathcal{P}$ has size $< B$):

1. For j from 1 to k:
   - Choose l_j uniformly at random from $L_j(\mathcal{P}) \setminus \{l_1, l_2, \ldots, l_{j-1}\}$.
2. Set $\mathcal{T}' \leftarrow \{T_{l_1}, T_{l_2}, \ldots T_{l_k}\}$.

Because the indices are arbitrary, $\mathbb{P}[\mathcal{T}' \text{ encloses } \mathcal{P}] = \mathbb{P}[\mathcal{T} \text{ encloses } \mathcal{P}] = c_i(\mathcal{P})$.

On the other hand, it is straightforward to show that the conditional probability $\mathbb{P}[\mathcal{T}' \text{ encloses } \mathcal{P} \mid \mathcal{T}]$ is exactly the product series $\bar{c}_i(\mathcal{P})$ (the j^{th} factor in the product series is the probability that $T_{l_j} > \tau_j$, conditioned on the fact that $T_{l_h} > \tau_h$ for all $h < j$).

Thus we have

$$\begin{aligned}\mathbb{E}[\bar{c}_i(\mathcal{P})] &= \mathbb{E}\left[\mathbb{P}[\mathcal{T}' \text{ encloses } \mathcal{P} \mid \mathcal{T}]\right] \\ &= \mathbb{P}[\mathcal{T}' \text{ encloses } \mathcal{P}] \\ &= c_i(\mathcal{P})\,.\end{aligned}$$

□