Restart Policies with Dependence among Runs: A Dynamic Programming Approach

Yongshao Ruan[1], Eric Horvitz[2], Henry Kautz[1]

[1] University of Washington, Seattle WA 98195, USA,
{ruan,kautz}@cs.washington.edu
[2] Microsoft Research, Redmond WA 98052, USA
horvitz@microsoft.com

Abstract. The time required for a backtracking search procedure to solve a problem can be reduced by employing randomized restart procedures. To date, researchers designing restart policies have relied on the simplifying assumption that runs are probabilistically independent from one another. We relax the assumption of independence among runs and address the challenge of identifying an optimal restart policy for the case of dependent runs. We show how offline dynamic programming can be used to generate an ideal restart policy, and how the policy can be used in conjunction with real-time observations to control the timing of restarts. We present results of experiments on applying the methods to create ideal restart policies for several challenging problems using two different solvers.

1 Introduction

Combinatorial search algorithms in many domains exhibit high variance in running time over fixed sets of problems [23, 6, 19, 13, 7, 24]. In some cases, the probability distribution for the running time of a search algorithm over a problem set is *heavy-tailed*, having *infinite* mean and variance [8, 10, 9]. Investigators have pursued an understanding of the basis for such great variation in run-time, and have sought to exploit the uncertainty in execution time to develop more predictable and efficient procedures [3, 12].

Several investigators have explored the value of *randomized restarts* [11]. In this area of work, randomness is overlayed on the branching heuristic of a systematic search algorithm. If the search algorithm does not terminate within some number of backtracks, referred to as a *cutoff*, the run is halted and the algorithm is restarted with a new *random seed*. The randomized restart method has been demonstrated to reduce the total execution time on a wide variety of problems in scheduling, theorem-proving, circuit synthesis, planning, and hardware verification [22].

In Horvitz *et al.* (2001) [14], we introduced a general paradigm for building probabilistic models that predict a search algorithm's performance on a given problem instance, on the basis of the algorithm's behavior during the first few search steps. We asserted that such Bayesian models provide a foundation for

work on speed-up learning and control of problem solvers. In other related work, in Kautz *et al.* (2002) [17], we showed that predictive models could be used to design superior dynamic restart strategies for randomized problem solvers, for the case where runs are probabilistically independent. In this paper, we extend these results to the more general and more complex situation where runs are probabilistically *dependent*, where each run can provide an update about the nature of the probability distribution that generated the problem instance.

We shall first review some details of prior work and define the problem of dynamic restarts with dependencies. We show that the optimal restart policy for the dependent case can be modeled as a dynamic programming (DP) problem. We present a method centering on the use of offline DP to generate an ideal restart policy. Then, we show how observations can be incorporated into the restart policy. Finally, we illustrate the efficacy of the restart policies with experiments on several representative problems.

2 Research on Restart Policies

The basis for using randomized restarts is straightforward: the longer a backtracking search algorithm runs without finding a solution, the more likely it is that the algorithm is exploring a barren part of the search space, rather than branching early on states of critical variables that will be necessary for a solution. The designers of restart policies must grapple with minimization of total run time given a tradeoff; as the cutoff time is reduced, the probability that any particular run will reach a solution is diminished, so runs become shorter but more numerous.

Previous theoretical work on the problem of determining an ideal cutoff has made two assumptions: first, that the only feasible observation is the *length* of a run; and second, that the runs are *independent*. Under these conditions Luby *et al.* [21] described provably optimal restart policies. In the case of complete knowledge of the distribution, the optimal policy is the fixed cutoff that minimizes $E(T_c)$, the expected time to solution restarting every c backtracks. In the case of no knowledge of the distribution, Luby *et al.* further showed that a universal schedule of cutoff values of the form

$$1, 1, 2, 1, 1, 2, 4, ...$$

gives an expected time to solution that is within a log factor of that given by the optimal fixed cutoff, and that no other universal schedule is better by more than a constant factor.

Although the results of Luby *et al.* were taken by many in the research community to have settled all open issues on restart strategies, many real-life scenarios violate both assumptions. In Horvitz *et al.* [14] and Kautz *et al.* [17] we demonstrated that features other than run time can be used in the restart control policy of backtrack solvers. We introduced a framework for constructing Bayesian models that can predict the run time of problem solvers, and showed that observations of various features capturing the state of the solver during the

first few steps of a run could be fused to predict the length of a run with a useful degree of accuracy. We described several approaches to apply the observation into restart control policies and showed that the dynamic restart policies with observations beat the static optimal policy of Luby *et al.* by 40% to 65%. However, these papers retained the limiting assumption that runs were probabilistically independent.

The assumption that runs are independent can be violated in a number of settings. Consider the case where we have knowledge of the existence of several different probability distributions over run time, $D_1, D_2, ...$ A problem instance is drawn from one of the distributions based on prior probabilities and each run is performed on that same instance. In this setting, observations about the time exhibited until a restart of one or more prior runs of the same instance provides a probabilistic update about the probability distribution over run time of current and future runs.

It is easy to see how restart policies for independent and dependent runs can differ. Consider the simple case of distributions based on point probabilities. Suppose in the independent case a run always ends in 10 or 100 steps with equal probability: $P(t_i = 10) = P(t_i = 100) = 0.5$. In this case the optimal policy is to always restart after 10 steps if the problem is not solved. On the other hand, consider the dependent case where in D_1 all runs take 10 steps and in D_2 all runs take 100 steps, and a fixed instance is chosen from one of the distributions with equal probability. Then the optimal policy is to run with a cutoff ≥ 100. If the problem is not solved after 10 steps then we know the problem *requires* 100 steps, so a solver should continue. Any fixed cutoff less than 100 gives a non-zero probability of never solving the problem—and thus an infinite expected run time.

Our paper addresses the challenge of designing restart policies for dependent runs. Our specific contributions include:

- Modeling the optimal restart policy for dependent runs as a DP problem
- Specification of different predictive models and showing how they are used in dynamic restart policies
- Evaluating the optimal restart policies empirically with and without run-time observations and comparing these results with the best fixed-cutoff policies and the universal restart policy of Luby *et al.*.

3 Dependent restarts without observations

To simplify the presentation, we will focus on dependent runs for the case of two run-time distributions (RTDs), which we shall denote as *source* distributions D_1 and D_2. The results can be extended in a straightforward manner to the case of n distributions. At the outset of problem solving, one of the distributions is chosen according to a given prior probability, but the choice is not revealed to the solver. For each run we specify a cutoff t, and a sample is drawn from the chosen RTD. If the sample's run time is less than t the problem is solved; otherwise, we perform another run with a new sample from the same distribution. Thus, with

each unsuccessful run, we gain additional information about the distribution that was initially chosen.

This analysis characterizes several problem-solving scenarios, including the scenarios where (i) each D_i is the RTD for a single instance under a randomized solver; (ii) each D_i corresponds to an ensemble of instances with similar RTD's; or (iii) each D_i is the RTD of a heterogeneous ensemble, and the solver gets a new problem instance for each run. Another scenario of interest is where each D_i corresponds to a heterogeneous ensemble of instances, and the same problem instance is used for each run. The analysis of this *single-instance* scenario requires additional mathematical machinery that relates the RTD of an ensemble to the RTD's of its individual instances under a randomized solver, as we will explain in a forthcoming paper.

If we had perfect knowledge of the distribution that generated the instance, the problem would collapse to the independent case described by Luby *et al.* Let us consider the case where we are uncertain about the source distribution. Our goal is to find the optimal policy $(t_1, t_2, ...)$, where t_k is the cutoff for k-th run, such that the total number of steps to a solution is minimized. After each unsuccessful run, the solver's beliefs about the source distribution should be updated. We shall first consider the situation where we are limited solely to evidence about the time spent on prior runs. We formulate the problem of finding the optimal restart policy for the dependent case as a Markov decision process (MDP) ([15]).

Formally, the dependent restart problem can be described as a Markov decision process as follows: Let d_i be the prior probability of a run being chosen from distribution D_i, $p_i(t)$ the probability that a run will be selected from D_i stopping exactly at t, and $q_i(t) = \sum_{t' \leq t} p_i(t')$ the cumulative function of $p_i(t)$, where $i = 1, 2$. We will always assume that p_i is non-trivial in the sense that $p_i(\text{inf}) < 1$. Each state is a tuple of (d_1, d_2) and the set of actions for all states is the set of all possible cutoffs. Given an action t (*i.e.*, cutoff = t) and state $S = (d_1, d_2)$, the next possible state is either that the problem is solved (the termination state), or $S' = (d_1', d_2')$, where d_1' and d_2' are the updated probabilities. Once the solver reaches the termination state, denoted $S_0 = (0, 0)$, it remains there at no further cost. An *optimal restart control policy* is one whose *expected cost* to reach the termination state is minimum.

Let T denote the event that a run has not found a solution in the cutoff t, and D_i denote the event that the instance is from distribution D_i, where $i = 1, 2$. In this analysis, we are only considering the prior run times in updating d_1 and d_2. So we have

$$\begin{aligned} d_i' &= \frac{P(T, D_i)}{\sum_{j=1,2} P(T, D_j)} \\ &= \frac{P(D_i)P(T|D_i)}{\sum_{j=1,2} P(D_j)P(T|D_j)} \\ &= \frac{d_i(1 - q_i(t))}{\sum_{j=1,2} d_j(1 - q_j(t))} \end{aligned}$$

where $i = 1, 2$.

The immediate cost of setting the cutoff to be t at state S (*i.e.*, the expected length of the run from state S with a cutoff t), denoted $R(S,t)$, is

$$\begin{aligned} R(S,t) &= \sum_{i=1,2} d_i (\sum_{t' \le t} t' p_i(t') + t(1 - q_i(t)) \\ &= \sum_{i=1,2} d_i (t - \sum_{t' < t} q_i(t')) \end{aligned} \tag{1}$$

Thus, the next state $S' = (d_1', d_2')$ depends only on the previous state $S = (d_1, d_2)$ and the corresponding cutoff t. The same is true for the immediate cost R. So the state space satisfies the Markov property. With this, finding the optimal restart policy for the dependent runs is an MDP.

Given a cutoff t, the transition probabilities from S to S' and the termination state S_0 are

$$\begin{aligned} P(S'|S,t) &= \sum_{i=1,2} d_i (1 - q_i(t)) \\ P(S_0|S,t) &= \sum_{i=1,2} d_i q_i(t) \end{aligned}$$

The transition probability from S to any state other than S' and S_0 is 0.

The optimal expected solution time from state $S = (d_1, d_2)$ is the optimized sum of the immediate cost $R(S,t)$ and the optimal expected solution time of the two possible future states, denoted $V^*(S)$, which is given by the following Bellman equation:

$$\begin{aligned} V^*(S) &= \min_t \{R(S,t) + P(S'|S,t) V^*(S') \\ &\quad + P(S_0|S,t) V^*(S_0)\} \\ &= \min_t \{R(S,t) + P(S'|S,t) V^*(S')\} \\ &= \min_t \{ \sum_{i=1,2} d_i (t - \sum_{t<t} q_i(t')) \\ &\quad + \sum_{i=1,2} d_i (1 - q_i(t)) V^*(S')\} \end{aligned}$$

where we use the relation of $V^*(S_0) = 0$ and Equation 1.

$V^*(S)$ can be computed using DP. We have experimented with both policy iteration and value iteration for DP. A restart policy is said to be *proper* if, for each state $S = (d_1, d_2)$, we only select a cutoff t such that $P(S_0|S,t) > 0$, *i.e.*, each state has a positive transition probability to the termination state. Both policy iteration and value iteration are proved to converge to the ideal optimal value in theory[2], with the assumption of proper restart policies. In our studies, we have found that policy iteration converges faster than value iteration. Thus, we choose policy iteration for the experiments presented below.

Another practical problem is that the state space is continuous and we thus potentially must solve the problem for an infinite state space. For computational tractability, we transform the continuous space into a discrete state space and then apply finite-state DP methods. We shall review the experiments for the case of dependent runs without observation in Section 5. Before reviewing these results, we shall explore the case where we consider observations gathered during run time, in addition to the time taken by previous runs.

4 Dependent restarts with observations

Beyond run time, other evidence about the behavior of a solver may be valuable for updating beliefs about the source distribution. Indeed, watching a trace or visualization of a backtracking search engine in action can provide updates about run time. As mentioned above, our earlier work provides a general framework for constructing Bayesian models to predict the run time of problem solvers, and shows how probabilistic models can be used to create optimal dynamic restart policies for the case of independent runs [14,17]. We now consider situations where the system can make observations that update beliefs about the current D_i.

Let us explore the case where an evidential feature F of the solver state is observed during a run. F can be taken to be a function of the initial trace of the solver, as calculated by a decision tree over low-level variables. F may be binary-valued—providing, for example, an update about whether the current run will last at least 10,000 steps or not. F can also be multi-valued; for example, the feature may indicate which leaf of predictive decision tree model should be used to infer the RTD associated with a set of features observed during the initial portion of the run. For simplicity of exposition, we shall consider the case where F is binary-valued, *i.e.*, 0 or 1.

Our analysis makes no assumptions about the *meaning* of F. In practice, we need to choose an F that helps us choose a better cutoff value for the run. One natural choice for F is the output of a decision tree that is trained to discriminate between instances from D_1 and D_2. We call such an F a *distribution predictor*. In the experiments below, we use a SAT/UNSAT distribution predictor which discriminates between satisfiable and unsatisfiable instances. Another natural choice is to base F on a decision tree trained to discriminate "short" (less than median run time) verses "long" (greater than median run time) runs from the prior-weighted union of D_1 and D_2. We call such an F a *run-time predictor*. Intuitively, a distribution predictor provides us with probabilistic information that can be used to tune the cutoff to be good for the predicted distribution, while a run-time predictor would allow us to discard runs that are predicted to be long. However, it is important to understand that the dynamic programming procedure for calculating the optimal sequence of cutoff values does not rely on the explicit semantics of the predictive models: it simply determines the optimal cutoffs for any specified predictor. In Section 5, we compare the use of SAT/USAT distribution and run-time predictors on a benchmark test suite.

Thus, in addition to the run-time evidence, we now include information about the observation of F, which we include in the extended definition of a state. The extended state can now be described as (d_1, d_2, F), where d_1, d_2 are the same as those described in previous section. We define the transition probability, state transition, and the corresponding cost as follows: Besides T and D_i defined above, let F_n denote the feature observed at nth run, where F_n can be 0 or 1.

Before we can derive the transition probabilities and state transitions, we must compute the *interim probability* $P(D_i, F_{n+1}|F_n, T)$. The interim probability is the probability that the instance is generated by distribution D_i and F_{n+1} is observed in the $(n+1)$-th run, given that the n-th run had observation F_n and there is no solution for t steps.

$$P(D_i, F_{n+1}|F_n, T) = P(F_{n+1}|F_n, T, D_i)P(D_i|F_n, T)$$

We obtain the first part of the right side of the above formula (the probability of observing F_{n+1} in the next run) by conditioning on the assumptions that the instance is from distribution D_i, F_n is observed in the n-th run, and no solution is found within t steps:

$$P(F_{n+1}|F_n, T, D_i) = P(F_{n+1}|D_i)$$

We assert that the probability of observation F is independent of the previous observations and the selected cutoff, conditioned on the assumption that the instance is drawn from D_i.

The second part of the right side of the formula, *i.e.*, $P(D = D_i|F_n, T)$, is the probability that the instance is from distribution D_i, assuming that F_n is observed and no solution is found within t steps.

$$\begin{aligned} P(D = D_i|F_n, T) &= \frac{P(T|F_n, D_i)P(D_i|F_n)}{P(T|F_n)} \\ &= \frac{d_i P(T|F_n, D_i)}{P(T|F_n)} \end{aligned}$$

Combining the two parts, we have

$$P(D_i, F_{n+1}|F_n, T) = \frac{d_i P(T|F_n, D_i)P(F_{n+1}|D_i)}{P(T|F_n)}$$

We note that $P(T|F_n, D_i)$, $P(F_{n+1}|D_i)$, and $P(T|F_n)$ can be derived from the data.

Thus, the transition probability from state $S_n = (d_1, d_2, F_n)$ with action t to $S_{n+1} = (d_1', d_2', F_{n+1})$ is

$$\begin{aligned} & P(S_{n+1}|S_n, T) \\ &= P(F_{n+1}|F_n, T) \\ &= \sum_{i=1,2} P(D_i, F_{n+1}|F_n, T) \\ &= \frac{\sum_{i=1,2} d_i P(T|F_n, D_i)P(F_{n+1}|D_i)}{P(T|F_n)} \end{aligned} \tag{2}$$

For the next state $S_{n+1} = (d'_1, d'_2, F_{n+1})$, we have

$$\begin{aligned} d'_i &= P(D_i|F_n, T, F_{n+1}) \\ &= \frac{P(D_i, F_{n+1}|F_n, T)}{P(F_{n+1}|F_n, T)} \\ &= \frac{d_i P(T|F_n, D_i) P(F_{n+1}|D_i)}{\sum_{i=1,2} d_i P(T|F_n, D_i) P(F_{n+1}|D_i)} \end{aligned}$$

With a binary-valued F, from state $S_n = (d_1, d_2, F_n)$ with cutoff $= t$, we know that a solver will be in one and only one of the three states: a solution is found and the solver will be in the termination state S_0, no solution is found but F is true, or no solution is found and F is false. Let $S'_{n+1} = (d'_1, d'_2, F_{n+1} = 1)$ and $S''_{n+1} = (d''_1, d''_2, F_{n+1} = 0)$ denote the last two states respectively. The transition probability from S_n to any other states is 0.

The immediate cost $R(S_n, t)$, *i.e.*, the expected length of the nth run, associated with setting the cutoff to t at state S_n, is given by Equation 1. Similar to no-observation analysis, the optimal expected solution time from state $S_n = (d_1, d_2, F_n)$, denoted $V^*(S_n)$, is the optimized sum of the immediate cost $R(S_n, t)$ plus the optimal expected solution time of the three possible future states, which is given by the following Bellman equation:

$$\begin{aligned} & V^*(S_n) \\ & = \min_t \{R(S_n, t) + P(S_0|S_n, t) V^*(S_0) \\ & + P(S'_{n+1}|S_n, t) V^*(S'_{n+1}) \\ & + P(S''_{n+1}|S_n, t) V^*(S''_{n+1})\} \\ & = \min_t \{R(S_n, t) + P(S'_{n+1}|S_n, t) V^*(S'_{n+1}) \\ & + P(S''_{n+1}|S_n, t) V^*(S''_{n+1})\} \\ & = \min_t \{R(S_n, t) + P(T)(P(S'_{n+1}|S_n, T) V^*(S'_{n+1}) \\ & + P(S''_{n+1}|S_n, T) V^*(S''_{n+1}))\} \\ & = \min_t \{\sum_{i=1,2} d_i (t - \sum_{t'<t} q_i(t')) \\ & + (1 - \sum_{i=1,2} d_i q_i(t))(P(S'_{n+1}|S_n, T) V^*(S'_{n+1}) \\ & + P(S''_{n+1}|S_n, T) V^*(S''_{n+1}))\} \end{aligned}$$

where $P(S'_{n+1}|S_n, T)$ and $P(S''_{n+1}|S_n, T)$ can be computed by Equation 2.

Similar to the case with no observation, the dynamic dependent restart policy for the case with run-time observations can be computed with the use of DP as described in Section 3.

5 Experiments and Results

We performed a set of experiments to explore our approach to finding optimal restart policies with and without observations. We investigated the dependent restart policies for the multiple-instance problem-solving scenario described as Case iii in Section 3. For the multiple-instance situation, we choose an instance from $D_i, i = 1, 2$ according to prior probabilities. Then, for each run, a new instance from the same distribution D_i is randomly selected. We can draw and attempt to solve as many instances as we would like, but the goal is to solve one instance as soon as possible.

We considered as benchmark domains the quasigroup completion problem [7, 1, 18] and the graph coloring problem. Experiments performed in earlier work by Horvitz *et al.* [14] only considered satisfiable problems. In the experiments performed for this paper, we considered distributions containing unsatisfiable as well as satisfiable problem instances. As described in Section 3, we investigated dependent restart policies for the multiple-instance problem-solving scenario (that is, a heterogeneous ensemble with a new instance drawn for each run). As we noted, a forthcoming paper will explore the more complex case of a fixed instance drawn from a heterogeneous ensemble.

5.1 Background on Benchmark Domains

A quasigroup is an ordered pair $(L, \cdot)$, where L is a set and $(\cdot)$ is a binary operation on L such that the equations $a \cdot x = b$ and $y \cdot a = b$ are uniquely solvable for every pair of elements a, b in L. The *order* N of the quasigroup is the cardinality of the set L. An *incomplete* or *partial Latin Square* P is a partially filled N by N table such that no symbol occurs twice in a row or a column. The Quasigroup Completion Problem (QCP) is the problem of determining whether the remaining entries of the table can be filled in such a way that we obtain a complete Latin Square. For our studies, we generated a total of 10,000 instances, of which 6,062 were satisfiable. The instances are of order 30 with 337 unassigned variables or "holes."

The second problem domain we explored is solving propositional satisfiability (SAT) encodings of the Graph Coloring Problem (GCP). Graph coloring problem is a well-known combinatorial problem from graph theory. Given a graph $G = (V, E)$, where V=$\{v_1, v_2, ..., v_n\}$ is the set of vertices and E the set of edges connecting the vertices, we seek to find a coloring $C : V \rightarrow N$, such that connected vertices always have different colors. The challenge is to decide whether a coloring of the given graph exists for a particular number of colors.

We use the following strategy for encoding graph coloring problem instances into SAT: Each assignment of a color to a single vertex is represented by a propositional variable; each coloring constraint (edge of the graph) is represented by a set of clauses ensuring that the corresponding vertices have different colors. Two additional sets of clauses ensure that valid SAT assignments assign exactly one color to each vertex. The instances used in our studies are generated using Culberson's flat graph generator [5]. The challenge is to decide whether the

instances are 3-colorable. The instances contain 5,000 satisfiable instances and 5,000 unsatisfiable instances. The instances are generated in such a way that all 3-colorable instances are 2-uncolorable and all 3-uncolorable instances are 4-colorable.

As a third domain, we explored a planning problem in the logistics domain. Kautz and Selman [16] showed that propositional SAT encodings of STRIPS-style planning problems could be efficiently solved by SAT engines. The logistics domain involves moving packages on trucks and airplanes among locations in different cities. In the logistics domain, a state is a particular configuration of packages and vehicles. We generated instances with 5 cities, 15 packages, 2 planes, and 1 truck per city. We generated a total of 7,900 instances, where 3,618 of the instances were satisfiable. To decrease the variance among instances, all of the satisfiable instances can be solved with 12 parallel steps but cannot be solved with 11 steps. All of the unsatisfiable instances cannot be solved with 12 steps but can be solved with 13 steps.

5.2 Learning Predictive Models

We used the Satz-Rand [11] randomized backtracking search engine for the problems encoded as SAT. Satz-Rand is a randomized version of the Satz system of Li and Anbulagan [20]. For QCP problems, we experimented with a specialized randomized CSP solver built using the ILOG constraint programming library.

We implemented the methods described by Horvitz *et al.* [14] to learn predictors for run time based on observations (the feature F described in Section 4). The solvers were instrumented so that low-level observational variables could be collected over an observational horizon of up to 100 solver choice points. (A *Choice point* is a state in the backtracking search procedure where the algorithm makes a variable assignment heuristically, rather than making an assignment that is forced via propagation of previously set values. Types of value propagation include unit propagation, backtracking, lookahead, and forward-checking. We employed Bayesian learning procedures developed by Chickering, Heckerman, and Meek [4] to induce predictors in the form of decision trees built from the summary statistics of the low-level variables, from training sets generated from approximately 10,000 runs. As mentioned in Section 4, we experimented with two kinds of predictors: SAT/UNSAT distribution predictors, classifying an instance as a satisfiable or unsatisfiable instance, and run-time predictors, classifying a run as short or long, depending on whether the run time was less than or greater than the median time for the training set. Details about the procedure for learning predictive models are described in [17].

5.3 Comparing Policies for Dynamic Dependent Restarts

As described earlier, the optimal restart policy for dependent runs can be constructed offline by using policy iteration for DP, where we transform the continuous and infinite state space into a discrete state space and then apply finite-state DP methods. For all of the experiments, we quantitized the search space

uniformly into 10,000 segments, taking into consideration the tradeoff between computational efficiency and accuracy. Policy construction via DP with policy iteration required about one hour on a Pentium-800 machine with 1 gigabyte of memory.

To characterize the improvements associated with the dynamic dependent restart policies, we ran comparative experiments with a fixed cutoff restart policy, where the same cutoff is used for every run. The *optimal fixed cutoff* restart policy selects the fixed cutoff which minimizes the expected solution time:[3]

$$\min_t \frac{d_i(t - \sum_{t'<t} q_i(t'))}{q_i(t)}$$

Beyond the case for the universal restart policy of Luby *et al.*[4], we constructed optimal restart policies from training data and tested the policies on test data that had not been used for training. Results comparing the optimal restart policy of the two predictive models and the optimal restart policy without observation with the best fixed cutoff restart policy are shown in Table 1. For the problem domains studied, we found that the expected run time of both of the dynamic restart policies with observation are lower than that of the optimal dynamic restart policy without observations. We attribute the improvement in solution time, ranging from about 10% to 30% for the domains, to effectively harnessing the predictive models for differentiating runs. For all of the problems, we found that the run-time predictive model yields faster solution times than the SAT/UNSAT distribution predictive model. We believe that this is based in differentiating short runs from long runs, which endows the solver with an ability to avoid expending time on non-promising long runs.

Restart Policy	QCP		Graph Coloring		Planning	
	ERT	Improvement(%)	ERT	Improvement(%)	ERT	Improvement(%)
Optimal, no predictor	33,895	86.8	45,960	87.4	25,948	79.5
Optimal, run-time	26,423	89.7	31,012	91.5	18,418	85.4
Optimal, distribution	26,564	89.6	36,272	90.0	23,724	81.2
Best fixed cutoff	33,926	86.8	48,276	86.7	26,058	79.4
Luby *et al.*universal	257,363	0	363,626	0	126,383	0

Table 1. Comparative results of optimal policies with and without observation with the best fixed cutoff and Luby *et al.*'s universal restart policy, where ERT is the expected run time (choice points) and improvements are measured over Luby *et al.*'s universal policy.

[3] Note that the cutoff is usually different from the optimal cutoffs for distributions D_1 and D_2.

[4] Luby *et al.*'s universal restart policy does *not* change from distribution to distribution.

6 Summary and Directions

We described the challenge of relaxing the assumption of independence in randomized restart procedures for backtracking search. We defined the dynamic dependent restart problem and showed how we can employ dynamic programming in offline procedures to generate ideal real-time restart policies. We first explored the case where a solver only considers information about the execution time of previous runs and then extended the analysis to include evidence about problem-solving behavior observed during runs. Finally, we presented the results of experiments in three domains, including quasigroup completion, graph coloring, and logistics-planning problems.

We are pursuing several extensions to the results presented here. Our ongoing work includes the development of dynamic dependent restart procedures for the single-instance scenario. In this setting, we seek to relate the RTD of an ensemble to the RTD's of its individual instances under a randomized solver. We are also studying the generalization of the methods to scenarios that make weaker assumptions about the nature of the underlying RTD. As part of this work, we are exploring the use of methods from reinforcement learning to infer the underlying distribution from previous search trajectories.

We believe that there is great opportunity in continuing to take a Bayesian perspective in tackling combinatorial problems, where we develop machinery and methods that allows solvers to be *believers* that take into consideration evidential observations about problem solving, and that can consider probabilistic dependencies among multiple solving sessions.

References

1. Dimitris Achlioptas, Carla P. Gomes, Henry A. Kautz, and Bart Selman. Generating satisfiable problem instances. In *AAAI/IAAI*, pages 256–261, 2000.
2. D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.
3. Hubie Chen, Carla Gomes, and Bart Selman. Formal models of heavy-tailed behavior in combinatorial search. *Lecture Notes in Computer Science*, 2239:408–??, 2001.
4. David Maxwell Chickering, David Heckerman, and Christopher Meek. A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of the Thirteenth Conference On Uncertainty in Artificial Intelligence (UAI-97)*, pages 80–89, Providence, RI, 1997. Morgan Kaufman Publishers.
5. Joseph C. Culberson and Feng Luo. Exploring the k-colorable landscape with iterated greedy. In David S. Johnson and Michael A. Trick, editors, *Dimacs Series in Discrete Mathematics and Theoretical Computer Science, Vol. 36*, pages 245–284, 1996.
6. I. Gent and T. Walsh. Easy Problems are Sometimes Hard. *Artificial Intelligence*, 70:335–345, 1993.

7. Carla P. Gomes and Bart Selman. Problem Structure in the Presence of Perturbations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 221–227, New Providence, RI, 1997. AAAI Press.
8. Carla P. Gomes, Bart Selman, and Nuno Crato. Heavy-tailed Distributions in Combinatorial Search. In Gert Smolka, editor, *Principles and practice of Constraint Programming (CP97) Lecture Notes in Computer Science*, pages 121–135, Linz, Austria., 1997. Springer-Verlag.
9. Carla P. Gomes, Bart Selman, Nuno Crato, and Henry Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. of Automated Reasoning*, 24(1–2):67–100, 2000.
10. Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, pages 431–438, New Providence, RI, 1998. AAAI Press.
11. Carla P. Gomes, Bart Selman, and Henry A. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.
12. Aaai-2000 workshop on leveraging probability and uncertainty in computation, 2000.
13. T. Hogg, B. Huberman, and C. Williams (Eds.). Phase Transitions and Complexity (Special Issue). *Artificial Intelligence*, 81(1–2), 1996.
14. Eric Horvitz, Yongshao Ruan, Carla Gomes, Henry Kautz, Bart Selman, and Max Chickering. A Bayesian approach to tackling hard computational problems. In *Proceedings the 17th Conference on Uncertainty in Artificial Intelligence (UAI-2001)*, pages 235–244, Seattle, USA, 2001.
15. R.A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.
16. H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1188–1194, Portland, OR, 1996. AAAI Press.
17. Henry Kautz, Eric Horvitz, Yongshao Ruan, Bart Selman, and Carla Gomes. Dynamic randomized restarts: Optimal restart policies with observation. To appear in AAAI, 2002.
18. Henry Kautz, Yongshao Ruan, D. Achlioptas, Carla P. Gomes, Bart Selman, and Mark Stickel. Balance and filtering in structured satisfiable problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, pages 351–358, 2001.
19. S. Kirkpatrick and B. Selman. Critical behavior in the satisfiability of random Boolean expressions. *Science*, 264:1297–1301, 1994.
20. Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 366–371. AAAI Pess, 1997.
21. M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Process. Letters*, pages 173–180, 1993.
22. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535, 2001.
23. B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In D. Johnson and M. Trick, editors, *Dimacs Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26*, pages 521–532. AMS, 1993.
24. T. Walsh. Search in a Small World. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1172–1177, Stockholm, Sweden, 1999.