Dynamic Restart Policies

Henry Kautz
University of Washington
kautz@cs.washington.edu

Eric Horvitz
Microsoft Research
horvitz@microsoft.com

Yongshao Ruan
University of Washington
ruan@cs.washington.edu

Carla Gomes
Cornell University
gomes@cs.cornell.edu

Bart Selman
Cornell University
selman@cs.cornell.edu

Abstract

We describe theoretical results and empirical study of context-sensitive restart policies for randomized search procedures. The methods generalize previous results on optimal restart policies by exploiting dynamically updated beliefs about the probability distribution for run time. Rather than assuming complete knowledge or zero knowledge about the run-time distribution, we formulate restart policies that consider real-time observations about properties of instances and the solver's activity. We describe background work on the application of Bayesian methods to build predictive models for run time, introduce an optimal policy for dynamic restarts that considers predictions about run time, and perform a comparative study of traditional fixed versus dynamic restart policies.

Introduction

The possibility of developing tractable approaches to combinatorial search has been a long-held goal in AI. We describe theoretical results on *dynamic restarts*, restart policies for randomized search procedures that take real-time observations about attributes of instances and about solver behavior into consideration. The results show promise for speeding up backtracking search—and thus, move us one step closer to tractable methods for solving combinatorial search problems.

Researchers have noted that combinatorial search algorithms in many domains exhibit a high degree of unpredictability in running time over any given set of problems (Selman, Kautz, & Cohen 1993; Gent & Walsh 1993; Kirkpatrick & Selman 1994; Hogg, Huberman, & Williams 1996; Gomes & Selman 1997; Walsh 1999). In the most extreme case, the running time of a search algorithm over a problem set is best modeled by a *heavy-tailed* (powerlaw) distribution, having *infinite* mean and/or variance (Gomes, Selman, & Crato 1997; Gomes, Selman, & Kautz 1998a; Gomes *et al.* 2000).[1] Investigators have sought to understand the basis for such great variation by modeling search as a process that generates self-similar or *fractal* trees (Smythe & Mahmound 1995). Research on *algorithm portfolios* and

on *randomized restarts* has shown that it is possible to develop more predictable and efficient procedures (Gomes & Hoos 2000) by minimizing the risk associated with committing large amounts of computation to instances that are likely to have long run times. In the first approach, a portfolio of search algorithms is executed in parallel. Experiments have shown that such portfolios may exhibit a low mean and low variance in run time, even if each member of the portfolio has high mean and variance (Gomes & Selman 2001). In the second method, randomness is added to the branching heuristic of a systematic search algorithm. If the search algorithm does not find a solution within a given number of backtracks, known as the *cutoff*, the run is terminated and the algorithm is restarted with a new random seed. Randomized restarts have been demonstrated to be effective for reducing total execution time on a wide variety of problems in scheduling, theorem proving, circuit synthesis, planning, and hardware verification (Luby, Sinclair, & Zuckerman 1993; Huberman, Lukose, & Hogg 1997; Gomes, Selman, & Kautz 1998b; Moskewicz *et al.* 2001).

In this paper, we extend prior results on *fixed* restart policies to more efficient *dynamic restarts* by harnessing predictive models to provide solvers with a real-time ability to *update beliefs* about run time. We first review previous work on restart policies. Then we review recent work on constructing Bayesian models that can be used to infer probability distributions over the run time of backtracking search procedures based on observational evidence. We introduce new results on optimal restart policies that consider observations about solver behavior. Finally, we demonstrate the efficacy of the restart policies with empirical studies of backtracking search for solving quasigroup, graph-coloring, and logistics-planning problems.

Research on Restart Policies

The basis for the value of 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 necessary for a solution. But when should the algorithm give up on a particular run and restart the execution after some randomization? 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.

[1]Technically, because real-world search spaces are large but finite, there must always be some upper bound on the running time. However, it is common practice to refer to such truncated heavy-tailed distributions simply as "heavy tailed," in the case where a heavy-tailed distribution fits the data over several orders of magnitude.

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 system has either *complete* knowledge or *no* knowledge of the run-time distribution of the solver on the given instance. Under these conditions Luby *et al.* (1993) described provably optimal restart policies. In the case of complete knowledge, 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, Luby 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 best fixed cutoff, and that no other universal schedule is better by more than a constant factor.

Although these results were taken by many in the research community to have settled all open issues on restart strategies, real-life scenarios typically violate both assumptions. For one, we often have *partial* knowledge of the run-time distribution of a problem solver. For example, consider the case of a satisfiability (SAT) solver running on a mix of satisfiable and unsatisfiable problem instances. In general, run-time distributions over satisfiable and unsatisfiable instances are quite different (Frost, Rish, & Vila 1997). We might know that a new given instance was drawn from one of several different distributions of satisfiable and unsatisfiable problems, but not know *which* distribution. We cannot calculate a fixed optimal cutoff value, but still wish to take advantage of the knowledge that we do have; the "log factor" of the simple universal schedule can be quite large in practice (two or more orders of magnitude including the constant factors).

The assumption that only the running time of the solver can be observed may also be violated. Beyond run time, other evidence about the behavior of a solver may be valuable for updating beliefs about the run-time distribution. Indeed, watching a trace or visualization of a backtracking search engine in action can be quite enlightening. Observers can watch the algorithm make a few bad variable assignments and then thrash for millions of steps in the "wrong" area of the search space. A person watching the system often has intuitions about when it might be best to restart the search.[2] Can a program also make such judgments? Can it recognize dynamically changing *patterns of activity* that indicate that the search is lost and would best be restarted?

Recently Horvitz *et al.* (2001), motivated by such questions—and the associated promise of developing sound dynamic restart policies—introduced a framework for constructing Bayesian models that can predict the run time of problem solvers. They showed that observations of various features over time capturing the trajectory of states of the solver during the first few seconds of a run could be fused to predict the length of a run with a useful degree of accuracy. They also sketched an approach to using learned predictive models to control the restart policy of the solver.

Our paper builds upon the framework of Horvitz *et al.* (2001) and presents theoretical and empirical results on optimal restart policies in the presence of observations about the state of a solver and partial knowledge of the run-time distribution. Our specific contributions include:

- Characterization of the knowledge conditions under which the runs of a solver are dependent or independent, and the impact this has on the nature of optimal restart policies;

- Specification of a class of provably optimal dynamic restart policies in the presence of solver-state observations;

- Empirical evaluation of these dynamic policies against the best fixed-cutoff policies; and

- An empirical study of the sensitivity of the predictive models to diminishing periods of observation, that shows that a surprisingly short period of observation is necessary to create accurate models.

Dependent and Independent Runs

Most work on restart strategies for backtracking search assume explicitly or implicitly that runs are *probabilistically independent* from one another; in the analyses, no information is considered to be carried over from one run to the next.[3] However, a careful analysis of informational relationships among multiple runs reveals that runs may be *dependent* in some scenarios: observing run i influences the probability distribution we assign to run $i + 1$.

Knowledge conditions under which the runs are independent include: (i) a new instance is drawn from a static ensemble of instances for each run, and the full run-time distribution for the ensemble is known; or (ii) some feature of each run can be observed that classifies the particular run as a random sample from a known run-time distribution D_i, regardless of whether the problem instance is fixed or changes with each run. By contrast, an example of dependent runs is when we know the run-time distributions of several different problem ensembles, a problem instance is drawn randomly from one of the ensembles, and each run is performed on that same instance. In this case, the failure of each run to find a solution within some cutoff changes our beliefs about which ensemble was selected.

The families of restart policies that are appropriate for the independent and dependent situations are distinct. Consider the simple case of identifying the best fixed cutoff policies where D_1 and D_2 are point probabilities. Suppose in the independent case a run always ends in 10 or 100 steps with equal probability: 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 an instance is chosen from one of the distributions. Then the best fixed-cutoff policy is to run with no cutoff, because a fixed cutoff of less than 100 gives a finite probability of never solving the problem.[4]

Independent Runs in Mixed Distributions

The restart policy for the case of independent runs in light of a single known probability distribution over run time is covered by Luby *et al.* (1993)'s results, as described above.

---

[2]Research in AI on restart strategies began with just such informal observations.

[3]An exception to this is recent work by di Silva on combining clause learning with restarts, where clauses learned in one run are carried over to the next run (Baptista & Marques-Silva 2000).

[4]In the general dependent case, optimal policies actually use a series of different cutoffs, as discussed in Ruan *et al.* (2002).

We consider, therefore, the case where each run is a random sample from one of a number of known run-time distributions, $D_1, D_2, \cdots, D_n$, where the choice of D_i is an independent event made according to some prior probability distribution.

If the system has no knowledge of which D_i is selected and makes no observations other than the length of the run, then this case also collapses to that handled by Luby *et al.* (1993):

Proposition 1 *The optimal restart policy for a mixed run-time distribution with independent runs and no additional observations is the optimal fixed cutoff restart policy for the combined distribution.*

It is more interesting, therefore, to consider situations where the system can make observations that update beliefs about the current D_i. Horvitz *et al.* (2001) segment observations into *static* and *dynamic* classes of evidence. Static observations are directly measurable features of a problem instance. As an example, one could measure the clause to variable ratio in a SAT instance, as has been long considered in work on random k-SAT (Mitchell *et al.* 1992; Selman & Kirkpatrick 1996). Dynamic features are measurements obtained via the process of problem solving; they are observations of a search algorithm's state while it is in the process of trying to solve a particular instance. Both kinds of evidence are useful for identifying the source distribution of an instance.

We shall simplify our analysis without loss of generality by considering a single evidential feature F that summarizes all observations made of the current instance/run pair. In the experiments described below F is a function of a decision tree over a set of variables that summarize the trace of the solver for the initial 1,000 steps of a run. The variables include the initial, final, average, and first derivatives of such quantities as the number of unassigned variables in the current subproblem, the number of unsatisfied constraints, the depth of the backtracking stack, and so on (see Horvitz *et al.* (2001) for a more detailed discussion for features and their use in probabilistic models of run time). The decision trees are created by labeling a set of test run traces as "long" or "short" relative to the median time to solution, and then employing a Bayesian learning procedure (Chickering, Heckerman, & Meek 1997) to build probabilistic dependency models that link observations to probability distributions over run time. Note that because the summary variables include some quantities that refer to the initial, unreduced problem (such as the initial number of unbound variables), the feature F combines static and dynamic observations.

The feature F may be binary-valued, such as whether the decision tree predicts that the current run will be longer than the median run time. In other experiments, described below, we define a multivalued F, where its value indicates the *particular leaf* of the decision tree that is reached when trace of a partial run is classified. In an ideal situation, F would indicate the D_i for which the current run is a random sample with perfect accuracy. Such an ideal F simplifies the analysis of optimal restart strategies, because we do not have to consider error terms. We can in fact achieve such perfect accuracy by a *resampling* technique, described below, whereby the F is used to *define* a set of distributions D_i (which in general are different from the distributions used to create the original decision tree). Therefore, without loss of generality, we will assume that F always indicates the actual D_i for the run.

Let us assume first that F is binary valued, so there are two distributions D_1 and D_2, and we wish to find the optimal restart policy. First we must decide what we mean by "optimal:" do we wish to minimize expected run time, minimize variance in run time, or some combination of both? In this paper, we pursue the minimization of expected run time, although in some applications one may be willing to trade off an increase in expected run time for a decrease in variance; such tradeoffs are discussed in work on algorithm portfolios (Huberman, Lukose, & Hogg 1997; Gomes, Selman, & Kautz 1998b).

Next, let us consider the *form* of the policy. Is the policy the same for every run, or can it evolve over time; that is, is the policy stationary? The assumption that the runs are independent immediately entails that the policy is indeed stationary as we do not learn anything new about the D_i over time. This is the key distinction between policies for independent and dependent restart situations. Therefore we can conclude that the policy must be a function of F alone:

Theorem 1 *In the case of independent runs, where the (only) observation F is made after T_0 steps during each run, and where F indicates whether a run is a member of D_1 or D_2, the optimal restart policy is either of the form:*

Set the cutoff to T_1 for a fixed $T_1 < T_0$.

or of the form:

Observe for T_0 steps and measure F;
If F is true then set the cutoff to T_1, else set the cutoff to T_2

for appropriate constants T_1, T_2.

The first case is the degenerate one where waiting to observe F is never helpful. In the second situation, we are able to take advantage of our prediction of how "lucky" the current run will be. In general, this kind of dynamic policy outperforms the optimal static policy where the observation is ignored. In fact, the dynamic policy can outperform the optimal static policy even if the optimal static cutoff is *less* than the time T_0 required to make an observation, if predictions are sufficiently accurate.

Optimal Dynamic Policies

What values should be chosen for T_1 and T_2? They are *not*, in general, the same as the optimal static cutoffs for the individual distributions. The optimal dynamic cutoff values are found by deriving an expression for the expected time to solution for any T_1 and T_2, and then selecting values that minimize the expression given the data available for D_1 and D_2.

Let us begin by reviewing the formula for the expected time to solution of a fixed cutoff policy for a single distribution. Let $p(t)$ be the probability distribution over a run stopping exactly at t, and $q(t) = \sum_{t' \leq t} p(t')$ be the cumulative probability distribution function of $p(t)$. For a given cutoff T, the expected number of runs required to find a solution is the mean of the Bernoulli distribution for independent trials with probability $q(T)$, $1/q(T)$. The expected length of each run is $(1 - q(T))T + \sum_{t \leq T} tp(t) = T - \sum_{t < T} q(t)$

(Luby, Sinclair, & Zuckerman 1993). Multiplying the expected time per run and the expected number of runs gives an expected time to solution of

$$E(T) = \frac{T - \sum_{t < T} q(t)}{q(T)} \qquad (1)$$

We can extend this result to the case of multiple distributions by considering the probability of different distributions and computing a new expectation. Taking d_i as the prior probability of a run being chosen from distribution D_i, $p_i(t)$ as the probability that a run selected from D_i will stop exactly at t, and $q_i(t)$ as the cumulative function of $p_i(t)$, the expected number of runs to find a solution using cutoff T_i, whenever a sample comes from D_i, is now $1/(\sum_i d_i q_i(T_i))$. The expected length of each run is $\sum_i d_i(T_i - \sum_{t < T_i} q_i(t))$ The product of these quantities yields the expected run time for a particular choice of T_i, and thus the optimal cutoff values are those that minimize this expectation.

Theorem 2 *For independent runs where each run is selected with probability d_i from known distribution D_i, the optimal dynamic restart policy uses cutoffs*

$$(T_1^*, \quad ..., \quad T_n^*) = \arg \min_{T_1,...,T_n} E(T_1, ... T_n) \qquad (2)$$

$$= \arg \min_{T_1,...,T_n} \frac{\sum_i d_i(T_i - \sum_{t < T_i} q_i(t))}{\sum_i d_i q_i(T_i)} \qquad (3)$$

If the search algorithm runs for at least T_0 steps to identify the relevant distribution D_i, then the optimal cutoffs are either uniformly some $T < T_0$, or are bounded below by T_0:

$$(T_1^*, ..., T_n^*) = \arg \min_{T_i \geq T_0} E(T_1, ... T_n) \qquad (4)$$

In the most general case, the set of T_i^* that minimizes the expected overall time to solution can be determined by a brute-force search over the empirical data. Beyond brute-force minimization, there is opportunity to use parameterized probability distributions to model the empirical data and to derive closed-form expressions for the T_i^*.

Optimal Pruning of Runs after Observation

An interesting special case of the optimal dynamic policy is the situation where the best action for one or more of the distributions is to restart immediately after observation. We wish to identify conditions where it is best to simply remove from consideration runs we determine to be "unlucky," following analysis of static features of the instance or some initial observation of the solver's behavior. We shall consider here the pruning conditions for the case of two distributions, based on properties of the distributions.

For a given cutoff T_1, we seek to identify the conditions under which $E(T_1, T_0 + \Delta)$ is never less than $E(T_1, T_0)$. By substituting in the formula for the expected time to solution (Equation 3) and performing some simplification, one can show that runs from D_2 should be pruned if, for all $\Delta > 0$, it is the case that:

$$\frac{\Delta - \sum_{T_0 \leq t < T_0 + \Delta} q_2(t)}{q_2(T_0 + \Delta) - q_2(T_0)} > E(T_1, T_0) \qquad (5)$$

The left-hand side of the inequality is the cost–benefit ratio for extending runs in D_2 following observation, and the right-hand side, representing the expected run time of the pruned policy, can be computed from the empirical data. An interesting feature of this formula is that d_1 and d_2 *disappear* from the left-hand side: the prior probabilities assigned to the two distributions are irrelevant.

Empirical Studies

We performed a set of empirical studies to explore the dynamic restart policies given evidence gathered about solver behavior. Our first benchmark domain was a version of the Quasigroup Completion Problem (QCP) (Gomes & Selman 1997). The basic QCP problem is to complete a partially-filled Latin square, where the "order" of the instance is the length of a side of the square. We used a version called *Quasigroup with Holes* (QWH), where problem instances are generated by erasing values from a completed Latin square. QWH problems are "balanced" if the same number of missing values appear in each row and column. QWH is NP-complete, and balanced QWH is the hardest known subset of QWH (Achlioptas *et al.* 2000; Kautz *et al.* 2001). Note that QWH problems are satisfiable by definition.

For the QWH domain, we experimented with both CSP and SAT (Boolean) problem encodings. The CSP solver was designed specifically for QWH and built using the ILOG constraint programming library. The CSP problems were (non-balanced) random QWH problems of order 34 with 380 unassigned holes (the hardest hole/order ratio for random problems). The SAT-encoded problems were solved with Satz-Rand (Gomes, Selman, & Kautz 1998b), a randomized version of the Satz system of Li and Anbulagan (Li & Anbulagan 1997). Satz implements the Davis-Putnam-Longemann-Loveland (DPLL) procedure with look-ahead and a powerful variable-choice heuristic. The SAT-encoded problems were balanced QWH problems of order 34 with 410 unassigned holes (the hardest hole/order ratio for balanced problems).

Following the cycle of experiments with QCP, we applied the methods to the propositional satisfiability (SAT) encodings of the Graph Coloring Problem (GCP) and Logistics Planning (LPlan) problems.

For the GCP domain, we experimented with the randomized SAT algorithm running on Boolean encodings. The instances used in our studies are generated using Culberson's flat graph generator (Culberson & Luo 1996). Each instance contains 450 vertices and 1,045 randomly generated edges. The challenge is to decide whether the instances are 3-colorable. The instances are generated in such a way that all 3-colorable instances are 2-uncolorable and all 3-uncolorable instances are 4-colorable. Half of the problems were 3-colorable (satisfiable in the Boolean encoding) and half were not (unsatisfiable).

For the LPlan domain, we again experimented with Satz-Rand algorithm running on Boolean encodings. Kautz and Selman (Kautz & Selman 1996) 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 between different 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, where the initial and goal placements

of packages was randomly determined. The parallel-plan length was fixed at 12 steps. To decrease the variance among instances, we filtered the output of the problem generator so that the satisfiable instances could be solved with 12 parallel steps but not 11 steps, and the unsatisfiable instances could not be solved with 12 steps but could be solved in 13 steps. As before, we selected half satisfiable and half unsatisfiable instances.

We implemented the methods described by Horvitz *et al.* (2001) to learn predictive models for run time for a problem solving scenario Horvitz *et al.* (2001) refer to as the *multiple-instance* problem. In multiple-instance problems, we draw instances from a distribution of instances and seek to solve *any* instance as soon as possible, or as many instances as possible for any amount of time allocated. For each case, we consider the states of multiple evidential variables observed during the observation horizon. In our experiments, observational variables were collected over an observational horizon of up to 1,000 solver choice points. *Choice points* are the states in search procedures where the algorithm assigns a value to variables where that assignment is not forced via propagation of previous set values. Such a situation occurs with unit propagation, backtracking, look-ahead, and forward-checking. At these points in problem solving a variable assignment is chosen according to the solver's particular heuristics.

For the studies described, we represented run time as a probability distribution over a binary variable with discrete states "short" versus "long." We defined short runs as cases completed before the median run time for the problem domain (see Table 1 for the median run times for each benchmark). As described in Horvitz *et al.* (2001), we employed Bayesian learning methods (Chickering, Heckerman, & Meek 1997) to generate graphical probabilistic models for solver run time. The resulting probabilistic graphical models, and associated decision trees that represent a compact encoding of the learned conditional probability distributions, are thus formulated to predict the likelihood of a run completing in the less than the median time, on the basis of observations of the beginning of the run.

Each of the training sets contained 2,500 runs (where each run is on a different instance), except for the QWH Boolean encoded problems, where the training set was of size 5,000. A separate test set of the same size as the training set for each domain was also created for the final evaluation of the different policies we considered.

Generating Distributions via Resampling

Using the inferred run-time distributions directly in our studies would imply an assumption that the model's inferences are accurate, gold-standard distributions. However, we know that the models are imperfect classifiers. The assumption of perfect model accuracy can be appropriately relaxed by overlaying additional error modeling. Such error modeling introduces terms that represent classification accuracy. To bypass the use of more cumbersome analyses that include a layer of error analysis, we instead performed resampling of the training data: we used the inferred decision trees to *define* different classes (representing specific sets of values of observed states of solver behavior), and relabeled the training data according to these classes. In other words, we use the branching structure of the decision trees—the leafs indicated by different sets of observations—to *define*

each sub-distribution $D_1, D_2, ...D_n$ and obtain statistics on each of these distributions by running the training data back through the decision tree, and computing the different run-time distributions for each value of F. Resampling the data to generate distributions lets us create distributions that encode predictive errors in an implicit manner.

Experiments with Dynamic Restarts

We performed experiments to compare dynamic restart policies with the fixed optimal restart policy of Luby *et al.* (1993). We considered two basic formulations of the dynamic restart policies, that we refer to as *binary* and *n-ary* policies. For the binary policy, runs are classified as either having *short* or *long* run-time distributions, based on the values of features observed during the observation phase. Runs from the training data are first bucketed into the different leafs of the decision tree based on the observed values of run-time observations. We define long and short distributions in terms of the decision-tree path indicated by the set of observations, asserting that all cases at a leaf of the decision tree that contains more than 60% of short runs are classified as a member of the short distribution, and otherwise as a member of the long distribution[5] For the *n-ary* policy, each leaf in the decision tree is considered as defining a *distinct* distribution.

For identifying the optimal set of cutoff–observation pairs in a dynamic restart policy, we used Eqn. 3 to search for the combination of cutoffs associated with minimum expected run time. We also considered a range of different observation periods, ranging from 10 to 1,000 choice points. The shortest window turned out to yield policies with the lowest expected run times; below we discuss specific results on the sensitivity of the policies to the length of the window.

For the binary dynamic restart case, the cutoff for the long distribution was found to be optimal, in both Satz and CSP experiments, when it is equal to the ideal observation horizon; thus, the optimization indicated that runs falling into the long distribution should be pruned in both of these cases. We confirmed the optimality of the pruning of the long distributions with the pruning condition specified by the inequality described in Eqn. 5.

For the *n-ary* restart case, we could not directly optimize the cutoffs with brute-force optimization, given the size of the decision trees. For example, in the Boolean-encoded QWH domain, the decision tree has 20 leafs, and in principle we would need to simultaneously vary 20 parameters. Therefore we simplified the search problem by pruning all runs that fell into leafs that contained less than 60% short runs, and then performing brute-force search to find the set of optimal cutoffs for the "short" leafs. Finally, we confirmed that pruning runs from the long leafs was indeed optimal by checking the pruning condition (Eqn. 5).

Beyond the dynamic policies and the fixed-optimal policy, we investigated for comparative purposes the time to solution with Luby's universal restart policy, and a *binary–naive* restart policy, composed by selecting distinct, separately optimal fixed cutoffs for the long and for the short distributions in the binary setting.

---

[5]Intuitively any threshold greater than 50% could be used. We empirically found for the benchmark domains studied that using a threshold of 60% gave the best performance for the restart policies that were ultimately formulated.
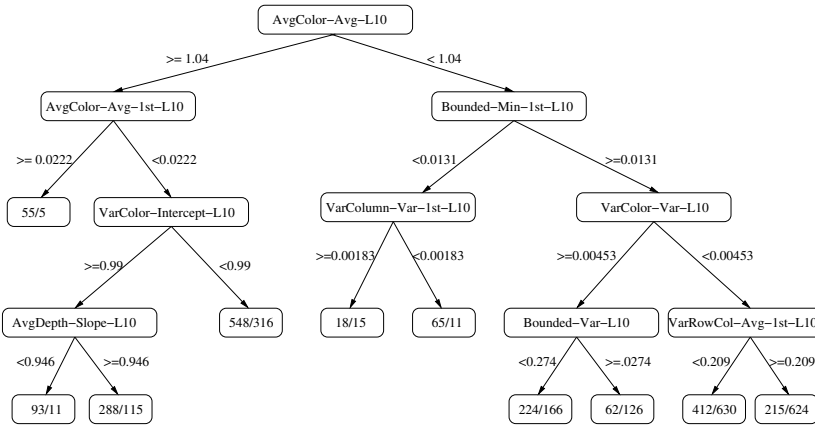
Figure 1: A predictive decision tree for a CSP QWH multiple-instance problem learned using an observation horizon of 10 choice points. Nodes represent observed features and arcs show the branching values. The number of cases of short versus long runs associated with the path are indicated at the leaves.
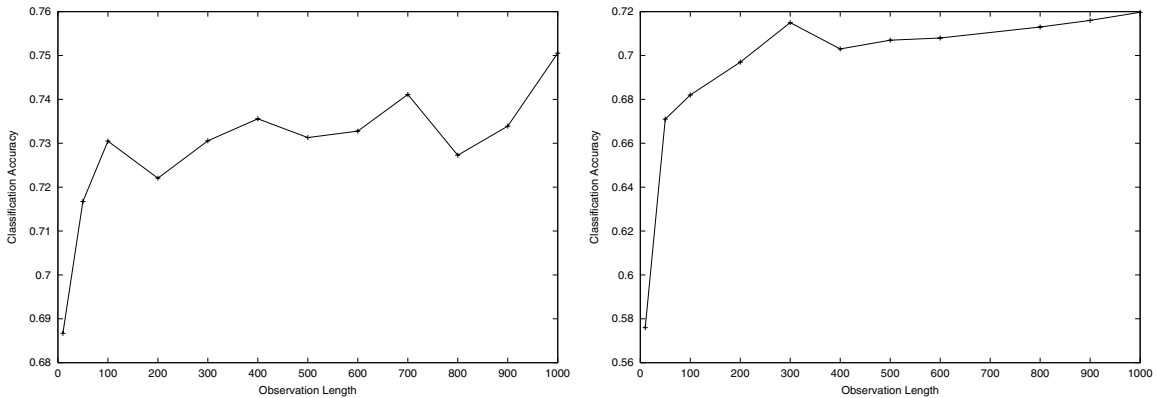

Figure 2: Analysis of the sensitivity of classification accuracy to the length of the observation horizon for CSP (left) and SATZ (right) on QWH multiple instances.

Sensitivity of Predictions to the Observation Horizon

As we highlighted in our earlier discussion of the pruning condition (Eqn. 5), the length of the observation window T_0 influences the optimal restart strategy. Long observation periods can limit the value of the dynamic methods by imposing a high constant tax whether or not an instance should be pruned immediately following the observation. Shorter horizons shift relationships and allow for a more efficient overall restart analysis; for example, a shorter horizon allows for the earlier pruning of runs following observations. Explicit knowledge of the relationship between observation horizon and accuracy thus provides another parameter for an optimization procedure to consider. Therefore, we studied the sensitivity of the predictive power of models to reduction of the observational horizon.

For the QWH domain, for both CSP and SATZ solvers, we collected run-time data for each instance over different observational horizons, varying from 10 to 1,000 choice points and constructed and tested the predictive power of distinct models. Fig. 1 displays an example of a learned decision trees for the CSP solver with observation horizon of 10 choice points. The internal nodes are labeled with the

name of the low-level solver-trace variable branched on. For example, along the left-hand branch we encounter the first two decision variables are:

- AvgColor-Avg-L10 — the average number of available colors for each empty square, averaged over the 10 choice points.

- AvgColor-Avg-1st-L10 — the first derivative of the average number of available colors for each empty square, measured at choice point 10.

The leaves are labeled with the number of short and long runs that appear there in the training data. For example, 93 short runs and 11 long runs reached the left-most leaf.

The learned predictive models for run time were found to overall show increasing classification accuracy with the use of longer observation horizons. Fig. 2 shows the sensitivity of the classification accuracy of the two solvers on the QWH multiple-instance problem to changes in the observation horizon. We found a steep increase in classification accuracy when the observation horizon is extended from the first 10 choice points to 100 choice points. Then the curve rises only slightly when the observation horizon is extended from 100 to 1,000 choice points. The sensitivity analysis

demonstrates that evidence collected in the first 100 choice yields the most discriminatory information. We believe that this finding makes intuitive sense; it is commonly believed that the first few choice points in the search of backtrack solvers have the most influence on the overall run time.

Despite the fact that predictive power was significantly better after 100 choice points than following 10 choice points, our optimization search over all possibly restart policies at different window sizes determined that the smallest window actually had the best cost/benefit ratio.[6] Thus, we used a window of 10 steps for the final experiments.

Results

After all the policies were constructed as described in the previous section, a fresh set of test data was used to evaluate each. The results, summarized in Table 1, were consistent across all of the problem domains: the dynamic n-ary policy is best, followed by the dynamic binary policy. We believe that the dominance of the n-ary dynamic restart policy over the binary dynamic policy is founded on the finer-grained, higher-fidelity optimization made possible with use of multiple branches of the decision trees. Improvements in solution times with the use of the dynamic policies range from about 40% to 65% over the use of Luby's fixed optimal restart policy.

The "naive" policy of using observations to predict the distribution for each run, and then using the optimal fixed cutoff for that distribution alone, performed poorly. This shows the importance of pruning long runs, and the value of the compute-intensive optimization of key parameters of the restart policy. Finally, the knowledge-free universal policy is about a factor of six slower than the best dynamic policy.

Summary and Directions

We introduced *dynamic restarts*, optimal randomized restart policies that take into consideration observations with relevance to run-time distributions. Our analysis included a consideration of key relationships among variables underlying decisions about pruning runs from consideration. To investigate the value of the methods, we performed several experiments that compare the new policies with static optimal restart procedures. To highlight the general applicability of the methods, studies were performed for quasigroup, graph coloring, and logistics planning problems.

We are pursuing several extensions to the results on dynamic restarts presented here. In one vein of work, we are exploring optimal randomized restart policies for the case of probabilistically *dependent* runs. As we noted above, with dependent runs, observations made in previously observed runs may influence the distribution over future runs. Dependent runs capture the situation where a solver performs restarts on the same instance. In this setting, observations about the time exhibited until a restart of one or more prior runs of the same instance can shift the probability distribution distribution over run time of current and future runs. Beyond dependent and independent runs, we are interested in policies for new kinds of challenges, representing mixes

of dependent and independent runs. For example, we are interested in the scenario where a solution can be generated either by continuing to restart a current instance until it is solved or by drawing a new instance from an ensemble. In another direction on generalization, we are exploring ensembles of instances containing satisfiable as well as unsatisfiable problems. In this work, we consider the likelihood of satisfiability in the analysis, given prior statistics and the updated posterior probabilities of satisfiability based on observations within and between runs. We are also exploring the use of richer observational models. Rather than rely on a single observational window, coupled with offline optimization, we are exploring the real-time adaptive control of when, how long, and which evidence is observed. As an example, our efforts on characterizing the sensitivity of predictive power of run-time predictions to the duration of the observation window suggest that the window might be controlled dynamically. In another thread of research, we are interested in leveraging inferences about run-time distributions to control search at a finer microstructure of problem solving. That is, we can move beyond the implicit restriction of being limited to the control of a parameter that dictates a cutoff time. We believe that reasoning about partial randomized restarts, using inferences about run time to guide decisions about backing up a solver to an intermediate state (rather than a complete restart) may lead to more flexible, efficient solvers.

We are excited about dynamic restart policies as representing a new class of procedures that tackle difficult combinatorial search problems via increased awareness of critical uncertainties and informational relationships. We hope that this work will stimulate additional efforts to integrate and harness explicit representations of uncertainty in methods for tackling difficult reasoning problems.

References

Achlioptas, D.; Gomes, C. P.; Kautz, H. A.; and Selman, B. 2000. Generating satisfiable problem instances. In *AAAI/IAAI*, 256–261.

Baptista, L., and Marques-Silva, J. P. 2000. The interplay of randomization and learning on real-world instances of satisfiability. In Gomes, C., and Hoos, H., eds., *Working notes of the AAAI-2000 Workshop on Leveraging Probability and Uncertatinty in Computation*. AAAI.

Chickering, D. M.; Heckerman, D.; and Meek, C. 1997. A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of the Thirteenth Conference On Uncertainty in Artificial Intelligence (UAI-97)*, 80–89. Providence, RI: Morgan Kaufman Publishers.

Culberson, J. C., and Luo, F. 1996. Exploring the k-colorable landscape with iterated greedy. In Johnson, D. S., and Trick, M. A., eds., *Cliques, Coloring and Satisfiability*, volume 26 of *DIMACS Series*. AMS. 245–284.

Frost, D.; Rish, I.; and Vila, L. 1997. Summarizing CSP hardness with continuous probability distributions. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 327–334. New Providence, RI: AAAI Press.

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

---

[6]We also experimented with a window of 0 steps—that is, using only static observations—but results were inconclusive. We are investigating the extent to which a carefully-chosen set of static features for a problem domain can match the performance of dynamic features for the multiple-instance case.

Restart Policy	Expected Runtime (Choice Points)			
	QCP (CSP)	QCP (Satz)	Graph Coloring (Satz)	Planning (Satz)
Dynamic n-ary (pruned)	3,295	8,962	9,499	5,099
Dynamic binary	5,220	11,959	10,157	5,366
Fixed optimal	6,534	12,551	13,894	6,402
Binary naive	17,617	12,055	14,669	6,962
Universal	12,804	29,320	38,623	17,359
Median (no cutoff)	69,046	48,244	39,598	25,255

Table 1: Comparative results of restart policies. Units are in choice points, which scales linearly with run time. The dynamic-binary policies pruned all runs classified as long. The fixed-optimal policy is that of Luby *et al.* (1993), based only on the complete run-time distribution without observations. The universal policy is Luby's log-optimal policy for unknown distributions. The binary-naive policy uses the decision tree to predict the distribution for the run, and then uses Luby's fixed optimal cutoff for that distribution.

Gomes, C., and Hoos, H. 2000. Aaai-2000 workshop on leveraging probability and uncertainty in computation.

Gomes, C. P., and Selman, B. 1997. Problem Structure in the Presence of Perturbations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, 221–227. New Providence, RI: AAAI Press.

Gomes, C. P., and Selman, B. 2001. Algorithm portfolios. *Artificial Intelligence* 126(1-2):43–62.

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

Gomes, C. P.; Selman, B.; and Crato, N. 1997. Heavy-tailed Distributions in Combinatorial Search. In Smolka, G., ed., *Principles and practice of Constraint Programming (CP97) Lecture Notes in Computer Science*, 121–135. Linz, Austria.: Springer-Verlag.

Gomes, C. P.; Selman, B.; and Kautz, H. 1998a. Boosting Combinatorial Search Through Randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 431–438. New Providence, RI: AAAI Press.

Gomes, C. P.; Selman, B.; and Kautz, H. A. 1998b. Boosting combinatorial search through randomization. In *AAAI/IAAI*, 431–437.

Hogg, T.; Huberman, B.; and Williams, C. 1996. Phase Transitions and Complexity (Special Issue). *Artificial Intelligence* 81(1–2).

Horvitz, E.; Ruan, Y.; Gomes, C.; Kautz, H.; Selman, B.; and Chickering, M. 2001. A Bayesian approach to tackling hard computational problems. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI-2001)*, 235–244. Morgan Kaufmann Publishers.

Huberman, B. A.; Lukose, R. M.; and Hogg, T. 1997. An economics approach to hard computational problems. *Science* 275(51).

Kautz, H., and Selman, B. 1996. Pushing the envelope: planning, propositional logic, and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1188–1194. Portland, OR: AAAI Press.

Kautz, H.; Ruan, Y.; Achlioptas, D.; Gomes, C. P.; Selman, B.; and Stickel, M. 2001. Balance and filtering in structured satisfiable problems. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-01)*.

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

Li, C. M., and Anbulagan. 1997. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 366–371. AAAI Pess.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of las vegas algorithms. *Information Process. Letters* 173–180.

Mitchell, D.; Selman, B.; ; and Levesque, H. 1992. Hard and easy distributions of sat problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, 459–465. AAAI Press.

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

Ruan, Y.; Horvitz, E.; and Kautz, H. 2002. Restart policies that consider dependence among runs: A dynamic programming approach. Submitted for publication.

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

Selman, B.; Kautz, H.; and Cohen, B. 1993. Local search strategies for satisfiability testing. In Johnson, D., and Trick, M., eds., *Dimacs Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26*. AMS. 521–532.

Smythe, R., and Mahmound, H. 1995. A survey of recursive trees. *Theoretical Probability and Mathematical Statistics* 51:1–27.

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