A Bayesian Approach to Tackling Hard Computational Problems (Preliminary Report) ⋆

Eric Horvitz

Microsoft Research, Redmond, WA 98052, USA

Yongshao Ruan

Dept. Comp. Sci. & Engr., Univ. of Washington, Seattle, WA 98195, USA

Carla Gomes

Dept. of Comp. Sci., Cornell Univ., Ithaca, NY 14853, USA

Henry Kautz

Dept. Comp. Sci. & Engr., Univ. of Washington, Seattle, WA 98195, USA

Bart Selman

Dept. of Comp. Sci., Cornell Univ., Ithaca, NY 14853, USA

Max Chickering

Microsoft Research, Redmond, WA 98052, USA

Abstract

We describe research and results centering on the construction and use of Bayesian models that can predict the run time of problem solvers. Our efforts are motivated by observations of high variance in the run time required to solve instances for several challenging problems. The methods have application to the decision-theoretic control of hard search and reasoning algorithms. We illustrate the approach with a focus on the task of predicting run time for general and domain-specific solvers on a hard class of structured constraint satisfaction problems. We describe the use of learned models to predict the ultimate length of a trial, based on observing the behavior of the search algorithm during an early phase of a problem session. Finally, we discuss how we can employ the models to inform dynamic run-time decisions.

Preprint submitted to Elsevier Science *15 May 2001*

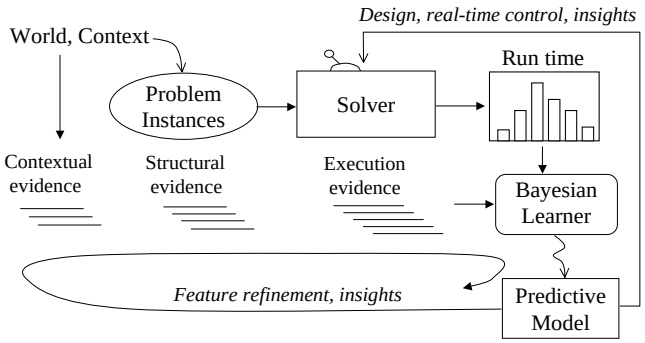

Fig. 1. Bayesian approach to problem solver design and optimization. We seek to learn predictive models to refine and control computational procedures as well as to gain insights about problem structure and hardness.

1 Introduction

The design of procedures for solving difficult problems relies on a combination of insight, observation, and iterative refinements that take into consideration the behavior of algorithms on problem instances. Complex, impenetrable relationships often arise in the process of problem solving, and such complexity leads to uncertainty about the basis for observed efficiencies and inefficiences associated with specific problem instances. We believe that recent advances in Bayesian methods for learning predictive models from data offer valuable tools for designing, controlling, and understanding automated reasoning methods.

Our overall methodology is highlighted in Fig. 1. As highlighted in the figure, the research is fundamentally iterative in nature; we exploit learning methods to identify and continue to refine observational variables and models, and attempt to use these models to enhance automated reasoning.

We focus on using machine learning to characterize variation in the run time of instances observed in inherently exponential search and reasoning problems. Predictive models for run time in this domain could provide the basis for more optimal decision making at the microstructure of algorithmic activity as well as inform higher-level policies that guide the allocation of resources.

We first provide background on the domain at the focus of our attention. Then, we describe our efforts to instrument problem solvers and to learn predictive models for run time. We describe the formulation of variables we used in learning and model construction and review the accuracy of the inferred

⋆ The final version of this paper appears in *Proc. Seventeenth Conference on Uncertainty in Artificial Intelligence (UAI 2001).*

models. Finally, we discuss opportunities for exploiting the models. We focus on the sample application of generating context-sensitive restart policies in randomized search algorithms.

2 Hard Search Problems

We have focused on applying learning methods to characterize run times observed in backtracking search procedures for solving NP-complete problems encoded as constraint satisfaction (CSP) and Boolean satisfiability (SAT). For these problems, it has proven extremely difficult to predict the particular sensitivities of run time to changes in instances, initialization settings, and solution policies. Numerous studies have demonstrated that the probability distribution over run times exhibit so-called *heavy-tails* [10]. Restart strategies have been used in an attempt to find settings for an instance that allow it to be solved rapidly, by avoiding costly journeys into a long tail of run time. Restarts are introduced by way of a parameter that terminates the run and restarts the search from the root with a new random seed after some specified amount of time passes, measured in choices or backtracks.

Progress on the design and study of algorithms for SAT and CSP has been aided by the recent development of new methods for generating hard random problem instances. Pure random instances, such as k-Sat, have played a key role in the development of algorithms for propositional deduction and satisfiability testing. However, they lack the structure that characterizes real world domains. Gomes and Selman [9] introduced a new benchmark domain based on *Quasigroups*, the Quasigroup Completion Problem (QCP). QCP captures the structure that occurs in a variety of real world problems such as timetabling, routing, and statistical experimental design.

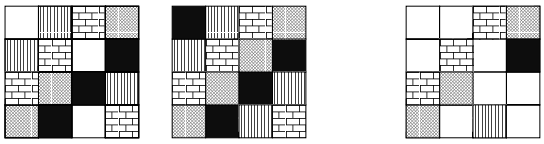

Fig. 2. Graphical representation of the quasigroup problem. Left: A quasigroup instance with its completion. Right: A balanced instance with two holes per row/column.

A quasigroup is a discrete structure whose multiplication table corresponds to a latin Square. A *latin square of order* n is an n x n array in which n distinct symbols are arranged so that each symbol occurs once in each row and column. A partial quaisgroup (or latin square) of order n is an n x n array based on n distinct symbols in which some cells may be empty but no

row or column contains the same element twice. The Quasigroup Completion Problem (QCP) can be stated as follows: *given a partial quasigroup of order n can it be completed to a quasigroup of the same order?*

QCP is an NP-complete problem [5] and random instances have been found to exhibit a peak in problem hardness as a function of the ratio of the number of uncolored cells to the total number of cells. The peak occurs over a particular range of values of this parameter, referred to as a region of *phase transition* [9,2]. A variant of the QCP problem, *Quasigroup with Holes* (QWH) [2], only includes satisfiable instances. The QWH generation procedure basically inverts the completion task: it *begins* with a randomly-generated completed Latin square, and then erases colors or "pokes holes." Completing QWH is NP-Hard [2]. A structural property that affects hardness of instances significantly is the pattern of the holes in row and columns. Balancing the number holes in each row and column makes the instances very hard [1].

3 Experiments with Problem Solvers

We performed a number of experiments with Bayesian feature selection and learning to elucidate previously hidden distinctions and relationships in SAT and CSP reasoners. We experimented with both a randomized SAT algorithm running on Boolean encodings of the QWH and a randomized CSP solver for QWH. The SAT algorithm was Satz-Rand [11], a randomized version of the Satz system of Li and Anbulagan [20]. Satz is the fastest known complete SAT algorithm for hard random 3-SAT problems, and is well suited to many interesting classes of structured satisfiability problems, including SAT encodings of quasigroup completion problems [10] and planning problems [17]. The solver is a version of the classic Davis-Putnam (DPLL) algorithm [7] augmented with one-step lookahead and a sophisticated variable choice heuristic. The lookahead operation is invoked at most choicepoints and finds any variable/value choices that would immediately lead to a contradiction after unit propagation; for these, the opposite variable assignment can be immediately made. The variable choice heuristic is based on picking a variable that if set would cause the greatest number of ternary clauses to be reduced to binary clauses. The variable choice-set was enlarged by a noise parameter of 30%, and value selection was performed deterministically by always branching on 'true' first.

The second backtrack search algorithm we studied is a randomized version of a specialized CSP solver for quasigroup completion problems, written using the ILOG solver constraint programming library. The backtrack search algorithm uses as variable choice heuristic a variant of the Brelaz heuristic. Furthermore, it uses a sophisticated propagation method to enforce the constraints that assert that all the colors in a row/column must be different. We refer to such a

constraint as *alldiff*. The propagation of the *alldiff* constraint corresponds to solving a matching problem on a bipartite graph using a network-flow algorithm [9,26,24].

We learned predictive models for run-time, motivated by two different classes of target problem. For the first class of problem, we assume that a solver is challenged by an instance and must solve that specific problem as quickly as possible. We term this the *Single-Instance* problem. In a second class of problem, we draw cases from a distribution of instances and are required to solve any instance as soon as possible, or as many instances as possible for any amount of time allocated. We call these challenges *Multiple-Instance* problems, and the subproblems as the *Any-Instance* and *Max-Instances* problems, respectively.

We collected evidence and built models for CSP and Satz solvers applied to the QWH problem for both the Single Instance and Multiple Instances challenge. We shall refer to the problem-solving experiments as CSP-QWH-Single, CSP-QWH-Multi, Satz-QWH-Single, and Satz-QWH-Multi. Building predictive Bayesian models for the CSP-QWH-Single and Satz-QWH-Single problems centered on gathering data on the probabilistic relationships between observational variables and run time for single instances with randomized restarts. Experiments for the CSP-QWH-Multi and Satz-QWH-Multi problems centered on performing single runs on multiple instances drawn from the same instance generator.

3.1 Formulating Evidential Variables

We worked to define variables that we believed could provide information on *problem-solving progress* for a period of observation in an early phase of a run that we refer to as the *observation horizon*. The definition of variables was initially guided by intuition. However, analysis of our early experiments helped us to refine sets of variables and to propose additional candidates. We limited features explored to those that could be computed with low (constant) overhead. We sought to collect information about base values as well as variants representing higher-level patterns that could be useful probes of the progress of the problem solver. For example, beyond exploring base observations about the program state at particular points in a case, we defined new families of observations such as first and second derivatives of the base variables.

Rather than include a separate variable in the model for each feature at each choicepoint—which would have led to an explosion in the number of variables and severely limited generalization—features and their dynamics were represented by variables for their summary statistics over the observation horizon.

The summary statistics included initial, final, average, minimum, and maximum values of the features during the observation period. For example, at each choice point, the SAT solver recorded the current number of binary clauses. The training data would thus included a variable for the average first derivative of the number of binary clauses during the observation period. Finally, for several of the features, we also computed a summary statistic that measured the number of times the sign of the feature changed from negative to positive or vice-versa.

We developed distinct sets of observational variables for the CSP and Satz solvers. The features for the CSP solver included some that were generic to any constraint satisfaction problem, such as the number of backtracks, the depth of the search tree, and the average domain size of the unbound CSP variables. Other features, such as the variance in the distribution of unbound CSP variables between different columns of the square, were specific to Latin squares. As we will see below, the inclusion of such domain-specific features were important in learning strongly predictive models. The CSP solver recorded 18 basic features at each choice point which were summarized by a total of 135 variables. The variables that turned out to be most informative for prediction are described in Sec. 4.1 below.

The features recorded by Satz-Rand were largely generic to SAT. We included a feature for the number of Boolean variables that had been set positively; this feature is problem specific in the sense that under the SAT encoding we used, only a *positive* Boolean variable corresponds to a *bound* CSP variable (*i.e.* a colored squared). Some features measured the current problem size (*e.g.* the number of unbound variables), others the size of the search tree, and still others the effectiveness of unit propagation and lookahead.

We also calculated two other features of special note. One was the logarithm of the total number of possible truth assignments (models) that had been ruled out at any point in the search; this quantity can be efficiently calculated by examining the stack of assumed and proven Boolean variable managed by the DPLL algorithm. The other is a quantity from the theory of random graphs called λ, that measures the degree of interaction between the binary clauses of the formula [23]. In all Satz recorded 25 basic features that were summarized in 127 variables.

3.2 Collecting Run-Time Data

For all experiments, observational variables were collected over an observational horizon of 1000 solver *choice points*. Choice points are states in search procedures where the algorithm assigns a value to variables where that assign-

ment is not forced via propagation of previous set values, as occurs with unit propagation, backtracking, lookahead, or forward-checking. These are point at which an assignment is chosen heuristically per the policies implemented in the problem solver.

For the studies described, we represented run time as a binary variable with discrete states short versus long. We defined short runs as cases completed before the median of the run times for all cases in each dataset. Instances with run times shorter than the observation horizon were not considered in the analyses.

4 Models and Results

We employed Bayesian model structure learning to infer predictive models from data and to identify key variables from the larger set of observations we collected. Over the last decade, there has been steady progress on methods for inferring Bayesian networks from data [6,27,12,13]. Given a dataset, the methods typically perform heuristic search over a space of dependency models and employ a Bayesian score to identify models with the greatest ability to predict the data. The Bayesian score estimates $p(model|data)$ by approximating $p(data|model)p(model)$. Chickering et al., developed methods for representing conditional probability distributions encoded in nodes in dependency models as compact decision graphs—a generalization of decision trees where non-root nodes may have more than one parent [4].

We employed the methods of Chickering et al. to infer models and to build decision graphs for run time from the data collected in experiments with CSP and Satz problem solvers applied to QWH problem instances. We shall describe sample results from the data collection and four learning experiments, focusing on the CSP-QWH-Single case in detail.

4.1 CSP-QWH-Single Problem

For a sample CSP-QWH-Single problem, we built a training set by selecting nonbalanced QWH problem instance of order 34 with 380 unassigned variables. We solved this instance 4000 times for the training set and 1000 times for the test data set, initiating each run with a random seed. We collected run time data and the states of multiple variables for each case over an observational horizon of 1000 choice points. We also created a marginal model, capturing the overall run-time statistics for the training set.

To avoid overfitting, we tuned a kappa parameter, used in the Bayesian score for penalizing complexity, by splitting the training set 70/30 into training and holdout data sets, respectively. We selected a kappa value by identifying a soft peak in the Bayesian score. This value was used to build a dependency model and decision graph for run time from the full training set. We then tested the abilities of the marginal model and the learned decision graph to predict the outcomes in the test data set. We computed a classification accuracy for the learned and marginal models to characterize the power of these models. The classification accuracy is the likelihood that the classifier will correctly identify the run time of cases in the test set. We also computed an average log score for the models.

Fig. 3 displays the learned Bayesian network for this dataset. The figure highlights key dependencies and variables discovered for the data set. Fig. 4 shows the decision graph for run time.

The classification accuracy for the learned model is 0.963 in contrast with a classification accuracy of 0.489 for the marginal model. The average log score of the learned model is -0.134 and the average log score of the marginal model was -0.693.

Because this was both the strongest and smallest model we learned, we will discuss the features it involves in more detail. Following Fig. 4 from left to right, these are:

VarRowCol measures the variance in the number of uncolored cells in the QWH instance across rows and across columns. A low variance indicates the open cells are evenly balanced thoughout the square. As noted earlier, balanced instances are harder to solve than unbalanced ones [1]. A rather complex summary statistic of this quantity appears at the root of the tree, namely the minimum of the first derivative of this quantity during the observation period. In future work we will be examining this feature carefully in order to determine why this particular statistic was most relevant.

AvgColumn measures the number of uncolored cells averaged across columns. A low value for this feature indicates the quasigroup is almost complete. The second node in the decision tree indicates that when the minimum value of this quantity across the entire observation period is low then the run is predicted to be short.

MinDepth is the minimum depth of all leaves of the search tree, and the summary statistic is simply the final value of this quantity. The third and fourth nodes of decision tree show that short runs are associated with high minimum depth and long runs with low minimum depth. This may be interpreted as indicating the search trees for the shorter runs have a more regular shape.

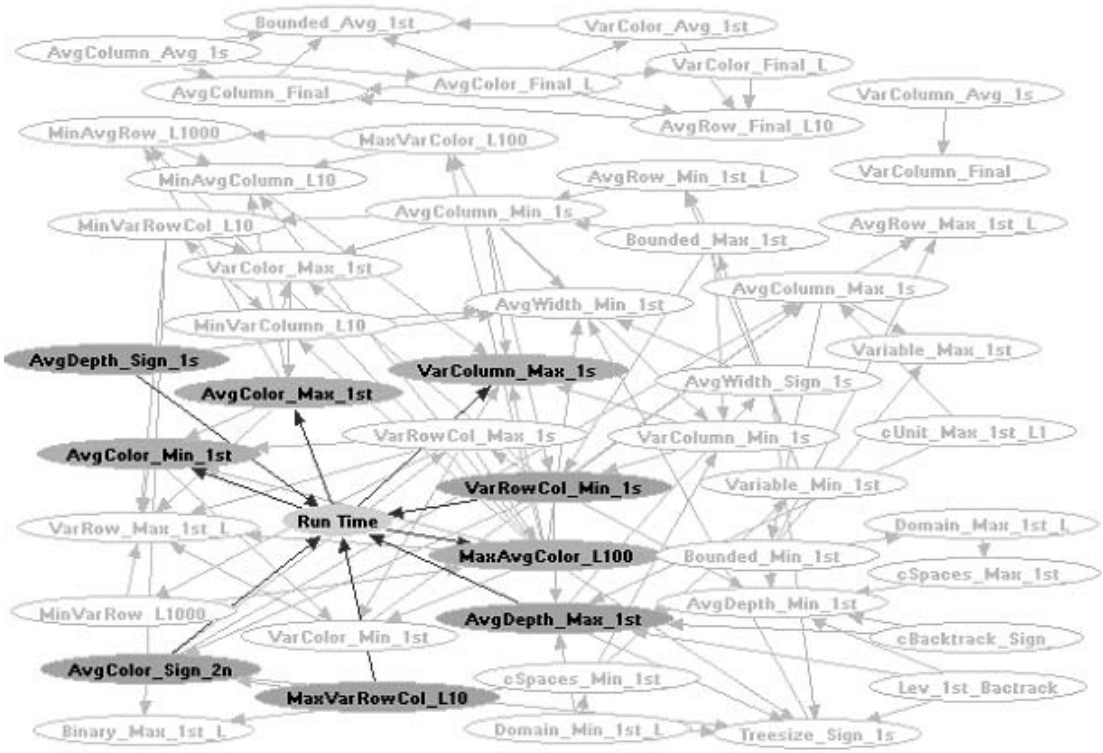

Fig. 3. The learned Bayesian network for a sample CSP-QWH-Single problem. Key dependencies and variables are highlighted.

AvgDepth is the average depth of a node in the search tree. The model discovers that short runs are associated with a high frequency in the change of the sign of the first derivative of the average depth. In other words, frequent fluctuations up and down in the average depth indicate a short run. We do not yet have an intuitive explanation for this phenomena.

VarRowColumn appears again as the last node in the decision tree. Here we see that if the maximum variance is low (*i.e.*, the problem remains balanced) then the run is long, as might be expected.

4.2 CSP-QWH-Multi Problem

For a CSP-QWH-Multi problem, we built training and test sets by selecting instances of nonbalanced QWH problems of order 34 with 380 unassigned variables. We collected data on 4000 instances for the training set and 1000 instances for the test set.

As we were running instances of potentially different fundamental hardnesses, we normalized the feature measurements by the size of the instance (measured in CSP variables) *after* the instances were initially simplified by forward-

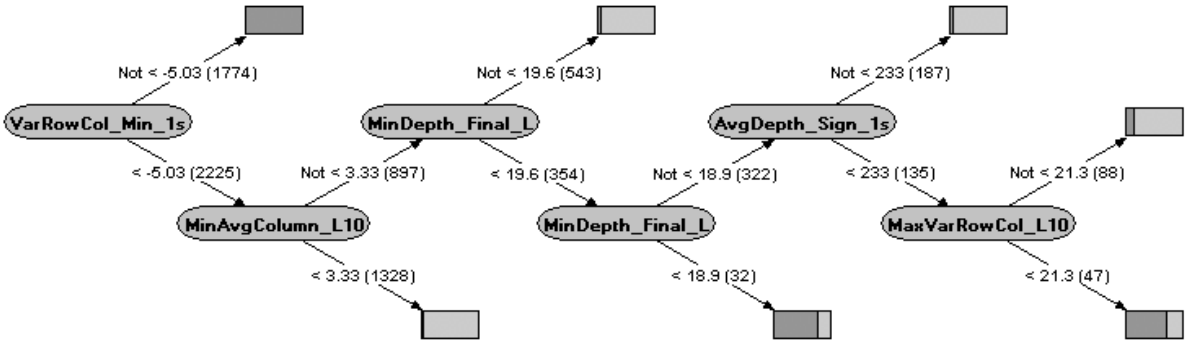

Fig. 4. The decision tree inferred for run time from data gathered in a CSP-QWH-Single experiment. The probability of a short run is captured by the light component of the bargraphs displayed at the leaves.

checking. That is, although all the instances originally had the same number of uncolored cells, polynomial time preprocessing fills in some of the cells, thus revealing the true size of the instance.

We collected run time data for each instance over an observational horizon of 1000 choice points. The learned model was found to have a classification accuracy of 0.715 in comparison to the marginal model accuracy of 0.539. The average log score for the learned model was found to be -0.562 and the average log score for the marginal model was -0.690.

4.3 Satz-QWH-Single Problem

We performed analogous studies with the Satz solver. In a study of the Satz-QWH-Single problem, we studied a single QWH instance (bqwh-34-410-16). We found that the learned model had a classification accuracy of 0.603, in comparison to a classification accuracy of 0.517 for the marginal model. The average log score of the learned model was found to be -0.651 and the log score of the marginal model was -0.693.

The predictive power of the SAT model was less than that of the corresponding CSP model. This is reasonable since the CSP model had access to features that more precisely captured special features of quasigroup problems (such as balance). The decision tree was still relatively small, containing 12 nodes that referred to 10 different summary variables.

Progress measures that turned out to be most relevant for the SAT model included:

- The maximum number of variables set to 'true' during the observation period. As noted earlier, this corresponds to the number of CSP variables that

would be bound in the direct CSP encoding.
- The number of models ruled out.
- The number of unit propagations performed.
- The number of variables eliminated by Satz's lookahead component: that is, the effectiveness of lookahead.
- The quantity λ described in Sec. 3.1 above, a measure of the constrainedness of the binary clause subproblem.

4.4 Satz-QWH-Multi Problem

For the experiment with the Satz-QWH-Multi problem, we executed single runs of QWH instances with the same parameters as the instance studied in the Satz-QWH-Single Problem (bqwh-34-410) for the training and test sets. Run time and observational variables were normalized in the same manner as for the CSP-QWH-Multi problem. The classification accuracy of the learned model was found to be 0.715. The classification accuracy of the marginal model was found to be 0.526. The average log score for the model was -0.557 and the average log score for the marginal model was -0.692.

5 Application: Dynamic Restart Policies

A predictive model can be used in many ways to control a solver. For example, the variable selection heuristic used to decompose the problem instance can be designed to minimize the expected solution time of the subproblems. Another application would involve building several models to predict the run time when different global strategies are used. As an example, we can learn to predict the relative performance of ordinary chronological backtrack search and dependency-directed backtracking with clause learning [16]. Such a predictive model could be used to decide whether the overhead of clause learning would be worthwhile for a particular instance.

Problem and instance-specific predictions of run time can also be used to drive dynamic cutoff decisions on when to suspend a current case and restart with a new random seed or new problem instance. Ideal policies that provide the fastest solutions to a particular instance or the highest rate of completed solutions can be formulated by computing the cumulative probability of achieving a solution as a function of the total dwell time allocated to an instance. Such information can be derived from our predictive models. In particular, for an ideal policy, we dynamically choose a cutoff time t that maximizes an expected rate of return of answers. Such a rate can be computed as the maximum ratio of the expected probability of solving an instance with increasing time and

the total time spent on the instance.

We can also use the predictive models to perform comparative analyses with previous policies. Luby *et al.* [21] have shown that the optimal restart policy, assuming full knowledge of the distribution, is one with a fixed cutoff. They also provide a universal strategy (using gradually increasing cutoffs) for minimizing the expected cost of randomized procedures, assuming no prior knowledge of the probability distribution. They show that the universal strategy is within a log factor of optimal. These results essential settle the distribution-free case.

Consider now the following dynamic policy: Observe a run for O steps. If a solution is not found, then predict whether the run will complete within a total of L steps. If the prediction is negative, then immediately restart; otherwise continue to run for up to a total of L steps before restarting if no solution is found.

An upper bound on the expected run of this policy can be calculated in terms of the model accuracy A and the probability P_i of a single run successfully ending in i or fewer steps. For simplicity of exposition we assume that the model's accuracy in predicting long or short runs is identical. The expected number of runs until a solution is found is $E(N) = 1/(A(P_L - P_O) + P_O)$. An upper bound on the expected number of steps in a single run can be calculated by assuming that runs that end within O steps take exactly O steps, and that runs that end in $O + 1$ to L steps take exactly L steps. The probability that the policy continues a run past O steps (*i.e.*, the prediction was positive) is $AP_L + (1 - A)(1 - P_L)$. An upper bound on the expected length of a single run is $E_{ub}(R) = O + (L - O)(AP_L + (1 - A)(1 - P_L))$. Thus, an upper bound on the expected time to solve a problem using the policy is $E(N)E_{ub}(R)$.

It is important to note that the expected time depends on both the accuracy of the model and the prediction point L; in general, one would want to vary L in order to optimize the solution time. Furthermore, in general, it would be better to design more sophisticated dynamic policies that made use of all information gathered over a run, rather than just during the first O steps. But even a non-optimized policy based directly on the models discussed in this paper can outperform the optimal fixed policy. For example, in the CSP-QWH-single problem case, the optimal fixed policy has an expected solution time of 38,000 steps, while the dynamic policy has an expected solution time of only 27,000 steps. Optimizing the choice of L should provide about an order of magnitude further improvement.

While it may not be surprising that a dynamic policy can outperform the optimal fixed policy, it is interesting to note that this can occur when the observation time O is *greater* than the fixed cutoff. That is, for proper values

of L and A, it may be worthwhile to observe each run for 1000 steps even if the optimal fixed strategy is to cutoff after 500 steps. These and other issues concerning applications of prediction models to restart policies are examined in detail in a forthcoming paper.

6 Related Work

Learning methods have been employed in previous research in a attempt to enhance the performance optimize reasoning systems. In work on "speed-up learning," investigators have attempted to increase planning efficiency by learning goal-specific preferences for plan operators [22,19]. Khardon and Roth explored the offline reformulation of representations based on experiences with problem solving in an environment to enhance run-time efficiency [18]. Our work on using probabilistic models to learn about algorithmic performance and to guide problem solving is most closely related to research on flexible computation and decision-theoretic control. Related work in this arena focused on the use of predictive models to control computation, Breese and Horvitz [3] collected data about the progress of search for graph cliquing and of cutset analysis for use in minimizing the time of probabilistic inference with Bayesian networks. The work was motivated by the challenge of identifying the ideal time for preprocessing graphical models for faster inference before initiating inference, trading off reformulation time for inference time. Trajectories of progress as a function of parameters of Bayesian network problem instances were learned for use in dynamic decisions about the partition of resources between reformulation and inference. In other work, Horvitz and Klein [14] monitored the progress of search in a propositional theorem prover and used measures of progress in updating the probability of truth or falsity of assertions. Stepping back to view the larger body of work on the decision-theoretic control of computation, measures of *expected value of computation* [15,8,25], employed to guide problem solving, rely on forecasts of the refinements of partial results with future computation. More generally, representations of problem-solving progress have been central in research on flexible or anytime methods—procedures that exhibit a relatively smooth surface of performance with the allocation of computational resources.

7 Future Work and Directions

This work represents a point in a space of ongoing research. We are pursuing additional modeling with an attempt to generalize across problems and problem classes. We are also pursuing the learning and use of models that

reason about run time at a finer grain. To date, we have used fixed policies over a short initial observation horizon. We are interested in pursuing longer observation horizons as well as using value of information to make dynamic decisions about the pattern and timing of monitoring. Finally, we investigating policies and methods for harnessing predictive models with a focus on optimizing restart policies.

8 Summary

We presented a methodology for characterizing the run time of problem instances for randomized backtrack-style search algorithms that have been developed to solve a hard class of structured constraint-satisfaction problems. The methods are motivated by recent successes with using fixed restart policies to address the high variance in running time typically exhibited by backtracking search algorithms. We described two distinct formulations of problem-solving goals and built probabilistic dependency models for each. Finally, we discussed opportunities for leveraging the predictive power of the models to optimize the performance of randomized search procedures by monitoring execution and dynamically setting cutoffs.

Acknowledgments

We thank Dimitris Achlioptas for his insightful contributions and feedback.

References

[1] D. Achlioptas, Carla P. Gomes, Henry Kautz, Yongshao Ruan, Bart Selman, and Mark Stickel. Balance and Filtering in Structured Satisfiable Problems. In *To Appear in: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*, Seattle, 2001.

[2] D. Achlioptas, Carla P. Gomes, Henry Kautz, and Bart Selman. Generating Satisfiable Instances. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-00)*, New Providence, RI, 2000. AAAI Press.

[3] J.S. Breese and E.J. Horvitz. Ideal reformulation of belief networks. In *Proceedings of UAI-90, Cambridge, MA*, pages 64–72. Assoc. for Uncertainty in AI, Mountain View, CA, July 1990.

[4] D.M. Chickering, D. Heckerman, and C. Meek. A Bayesian approach to learning Bayesian networks with local structure. In *Proceedings of UAI-97,* Providence, RI, pages 80–89. Morgan Kaufmann, August 1997.

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

[6] G. Cooper and E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.

[7] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1979.

[8] T.L. Dean and M.P. Wellman. *Planning and Control*, chapter 8.3 Temporally Flexible Inference, pages 353–363. Morgan Kaufmann, San Francisco, 1991.

[9] Carla P. Gomes and Bart Selman. Problem Structure in the Presence of Perturbations. In *Proceedings of AAAI-97*, New Providence, RI, 1997. AAAI Press.

[10] 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.

[11] 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)*, New Providence, RI, 1998. AAAI Press.

[12] D. Heckerman, J. Breese, and K. Rommelse. Decision-theoretic troubleshooting. *CACM*, 38:3:49–57, 1995.

[13] D. Heckerman, D.M. Chickering, C. Meek, R. Rounthwaite, and C. Kadie. Dependency networks for density estimation, collaborative filtering, and data visualization. In *Proceedings of UAI-2000,* Stanford, CA, pages 82–88. 2000.

[14] E. Horvitz and A. Klein. Reasoning, metareasoning, and mathematical truth: Studies of theorem proving under limited resources. In *Proceedings of UAI-95*, pages 306–314, Montreal, Canada, August 1995. Morgan Kaufmann, San Francisco.

[15] E.J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of AAAI-88*, pages 111–116. Morgan Kaufmann, San Mateo, CA, August 1988.

[16] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *AAAI/IAAI*, pages 203–208. AAAI Press, 1997.

[17] Henry Kautz and Bart Selman. Unifying Sat-based and Graph-based Planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, Stockholm, 1999.

[18] R. Khardon and D. Roth. Learning to reason. In *In Proceedings of AAAI-94*, pages 682–687. AAAI Press, July 1994.

[19] C. Knoblock, S. Minton, and O. Etzioni. Integration of abstraction and explanation-based learning in prodigy. In *Proceedings of AAAI-91.* AAAI, Morgan Kaufmann, August 1991.

[20] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the IJCAI-97*, pages 366–371. AAAI Pess, 1997.

[21] M. Luby, A. Sinclair, and D. Zuckerman. Optimal speedup of Las Vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

[22] S. Minton, J. G. Carbonell, C. A. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40:63–118, 1989.

[23] M. Molloy, R. Robinson, H. Robalewska, and N. Wormald. Factorisations of random regular graphs.

[24] Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of The Twelfth National Conference on Artificial Intelligence*, pages 362–367, 1994.

[25] S. Russell. Fine-grained decision-theoretic search control. In *Proceedings of Sixth Conference on Uncertainty in Artificial Intelligence*. Assoc. for Uncertainty in AI, Mountain View, CA, August 1990.

[26] P. Shaw, K. Stergiou, and T. Walsh. Arc Consistency and Quasigroup Completion. In *Proceedings of the ECAI-98 workshop on non-binary constraints*, 1998.

[27] D. Spiegelhalter, A. Dawid, S. Lauritzen, and R. Cowell. Bayesian analysis in expert systems. *Statistical Science*, (8):219–282, 1993.