Monitoring the Progress of Anytime Problem-Solving

Eric A. Hansen and Shlomo Zilberstein
Computer Science Department
University of Massachusetts
Amherst, MA 01003
{hansen,shlomo}@cs.umass.edu

Abstract

Anytime algorithms offer a tradeoff between solution quality and computation time that has proved useful in applying artificial intelligence techniques to time-critical problems. To exploit this tradeoff, a system must be able to determine the best time to stop deliberation and act on the currently available solution. When the rate of improvement of solution quality is uncertain, monitoring the progress of the algorithm can improve the utility of the system. This paper introduces a technique for run-time monitoring of anytime algorithms that is sensitive to the variance of the algorithm's performance, the time-dependent utility of a solution, the ability of the run-time monitor to estimate the quality of the currently available solution, and the cost of monitoring. The paper examines the conditions under which the technique is optimal and demonstrates its applicability.

Introduction

Anytime algorithms are being used increasingly for time-critical problem-solving in domains such as planning and scheduling (Boddy & Dean 1994; Zilberstein 1996), belief network evaluation (Horvitz, Suermondt, & Cooper 1989; Wellman & Liu 1994), database query processing (Shekhar & Dutta 1989; Smith & Liu 1989), and others. The defining property of an anytime algorithm is that it can be stopped at any time to provide a solution, such that the quality of the solution increases with computation time. This property allows a tradeoff between computation time and solution quality, making it possible to compute approximate solutions to complex problems under time constraints. It also introduces a problem of meta-level control: making an optimal time/quality tradeoff requires determining how long to run the algorithm, and when to stop and act on the currently available solution.

Meta-level control of an anytime algorithm can be approached in two different ways. One approach is to allocate the algorithm's running time before it starts and to let the algorithm run for the predetermined length of time no matter what (Boddy & Dean 1994). If there is little or no uncertainty about the rate of improvement of solution quality, or about how the urgency for a solution might change after the start of the algorithm, then this approach can determine an optimal stopping time. Very often, however, there is uncertainty about one or both. For AI problem-solving in particular, variance in solution quality is common (Paul et al. 1991). Because the best stopping time will vary with fluctuations in the algorithm's performance, a second approach to meta-level control is to monitor the progress of the algorithm and to determine at run-time when to stop deliberation and act on the currently available solution (Breese & Horvitz 1991; Horvitz 1990; Zilberstein & Russell 1995).

Monitoring the progress of anytime problem-solving involves assessing the quality of the currently available solution, making revised predictions of the likelihood of further improvement, and engaging in metareasoning about whether to continue deliberation. Previous schemes for run-time monitoring of anytime algorithms have assumed continuous monitoring, but the computational overhead this incurs can take resources away from problem-solving itself. This paper introduces a framework in which the run-time overhead for monitoring can be included in the problem of optimizing the stopping time of anytime problem-solving. It describes a framework for determining not only when to stop an anytime algorithm, but at what intervals to monitor its progress and re-assess whether to continue or stop. This framework makes it possible to answer such questions as:

- How much variance in the performance of an anytime algorithm justifies adopting a run-time monitoring strategy rather than determining a fixed running time ahead of time?
- How should the variance of an algorithm's performance affect the frequency of monitoring?
- Is it better to monitor periodically or to monitor more frequently toward the algorithm's expected stopping time?

For a large class of problems, the rate of improvement of solution quality is the only source of uncertainty about how long to continue deliberation. Examples include optimizing a database query (Shekhar & Dutta 1989), reformulating a belief net before solving it (Breese & Horvitz 1991), and planning the next move in a chess game (Russell & Wefald 1991). For other problems, the utility of a solution may also depend on the state of a dynamic environment that can change unpredictably after the start of the algorithm. Examples include real-time planning and diagnosis (Boddy & Dean 1994; Horvitz 1990). For such problems, meta-level control can be further improved by monitoring the state of the environment as well as the progress of problem-solving. We focus in this paper on uncertainty about improvement in solution quality. However, the framework can be extended in a reasonably straightforward way to deal with uncertainty about the state of a dynamic environment.

We begin by describing a framework for constructing an optimal policy for monitoring the progress of anytime problem-solving, assuming the quality of the currently available solution can be measured accurately at run-time. Because this assumption is often unrealistic, we then describe how to modify this framework when a run-time monitor can only estimate solution quality. A simple example is described to illustrate these results. The paper concludes with a brief discussion of the significance of this work and possible extensions.

Formal Framework

Meta-level control of an anytime algorithm – deciding how long to run the algorithm and when to stop and act on the currently available solution – requires a model of how the quality of a solution produced by the algorithm increases with computation time, as well as a model of the time-dependent utility of a solution. The first model is given by a *performance profile* of the algorithm. A conventional performance profile predicts solution quality as a function of the algorithm's overall running time. This is suitable for making a one-time decision about how long to run an algorithm, before the algorithm starts. To take advantage of information gathered by monitoring its progress, however, a more informative performance profile is needed that makes it possible to predict solution quality as a function of both time allocation and the quality of the currently available solution.

Definition 1 *A* ***dynamic performance profile*** *of an anytime algorithm, $Pr(q_j|q_i, \Delta t)$, denotes the probability of getting a solution of quality q_j by resuming the algorithm for time interval Δt when the currently available solution has quality q_i.*

We call this conditional probability distribution a dynamic performance profile to distinguish it from a performance profile that predicts solution quality as a function of running time only. The conditional probabilities are determined by statistical analysis of the behavior of the algorithm. For simplicity, we rely on discrete probability distributions. Time is discretized into a finite number of time steps, $t_0 \ldots t_n$, where t_0 represents the starting time of the algorithm and t_n its maximum running time. Similarly, solution quality is discretized into a finite number of levels, $q_0 \ldots q_m$, where q_0 is the lowest quality level and q_m is the highest quality level. Let q_{start} denote the starting state of the algorithm before any result has been computed. By discretizing time and quality, the dynamic performance profile can be stored as a three-dimensional table; the degree of discretization controls a tradeoff between the precision of the performance profile and the size of the table needed to store it. A dynamic performance profile can also be represented by a compact parameterized function.

Meta-level control requires a model of the time-dependent utility of a solution as well as a performance profile. We assume that this information is provided to the monitor in the form of a time-dependent utility function.

Definition 2 *A* ***time-dependent utility function,*** *$U(q_i, t_k)$, represents the utility of a solution of quality q_i at time t_k.*

In this paper, we assume that utility is a function of time and not of an external state of the environment. This assumption makes it possible to set to one side the problem of modeling uncertainty about the environment in order to focus on the specific problem of uncertainty about improvement in solution quality.

Finally, we assume that monitoring the quality of the currently available solution and deciding whether to continue or stop incurs a cost, C. Because it may not be cost-effective to monitor problem-solving continuously, an optimal policy must specify when to monitor as well as when to stop and act on the currently available solution. For each time step t_k and quality level q_i, the following two decisions must be specified:

1. how much additional time to run the algorithm; and
2. whether to monitor at the end of this time allocation and re-assess whether to continue, or to stop without monitoring.

Definition 3 *A* ***monitoring policy,*** *$\pi(q_i, t_k)$, is a mapping from time step t_k and quality level q_i into a monitoring decision $(\Delta t, m)$ such that Δt represents the additional amount of time to allocate to the anytime algorithm, and m is a binary variable that represents whether to monitor at the end of this time allocation or to stop without monitoring.*

An initial decision, $\pi(q_{start}, t_0)$, specifies how much time to allocate to the algorithm before monitoring for the first time or else stopping without monitoring. Note that the variable Δt makes it possible to control the time interval between one monitoring action and the next; its value can range from 0 to $t_n - t_i$, where t_n is the maximum running time of the algorithm and t_i is how long it has already run. The binary variable m makes it possible to allocate time to the algorithm without necessarily monitoring at the end of the time interval; its value is either *stop* or *monitor*.

An *optimal monitoring policy* is a monitoring policy that maximizes the expected utility of an anytime algorithm.

Given this formalization, it is possible to use dynamic programming to compute a combined policy for monitoring and stopping. Dynamic programming is often used to solve optimal stopping problems; the novel aspect of this solution is that dynamic programming is also used to determine when to monitor. A monitoring policy is found by optimizing the following value function:

$$V(q_i, t_k) = \max_{\Delta t, m} \left[\begin{array}{l} \sum_j Pr(q_j | q_i, \Delta t) U(q_j, t_k + \Delta t) \\ \text{if } m = \text{stop}, \\ \sum_j Pr(q_j | q_i, \Delta t) V(q_j, t_k + \Delta t) - C \\ \text{if } m = \text{monitor} \end{array} \right.$$

Theorem 1 *A monitoring policy that maximizes the above value function is optimal when quality improvement satisfies the Markov property.*

This is an immediate outcome of the application of dynamic programming under the Markov assumption (Bertsekas 1987). The assumption requires that the probability distribution of future quality depends only on the current "state" of the anytime algorithm, which is taken to be the quality of the currently available solution. The validity of this assumption depends on both the algorithm and how solution quality is defined, and so must be evaluated on a case-by-case basis. But we believe it is at least a useful approximation in many cases.

Uncertain measurement of quality

We have described a framework for computing a policy for monitoring an anytime algorithm, given a cost for monitoring. Besides the assumption that quality improvement satisfies the Markov property, the optimality of the policy depends on the assumption that the quality of the currently available solution can be measured accurately by a run-time monitor. How reasonable is this second assumption likely to be in practice?

We suggest that for certain types of problems, calculating the precise quality of a solution at run-time is not feasible. One class of problems for which anytime algorithms are widely used are optimization problems in which a solution is iteratively improved over time by minimizing or maximizing the value of an objective function. For such problems, the quality of an approximate solution is usually measured by how close the approximation comes to an optimal solution. For cost-minimization problems, this can be defined as

$$\frac{Cost(\text{Approximate Solution})}{Cost(\text{Optimal Solution})}$$

The lower this *approximation ratio*, the higher the quality of the solution, and when it is equal to one the solution is optimal.

The problem with using this measure of solution quality for run-time monitoring is that it requires knowing the optimal solution at run-time. This is no obstacle to using it to construct a performance profile for an anytime algorithm, because the performance profile can be constructed off-line and the quality of approximate solutions measured in terms of the quality of the eventual optimal solution. But a run-time monitor needs to make a decision based on the approximate solution currently available, without knowing what the optimal solution will eventually be. As a result, it cannot know with certainty the actual quality of the approximate solution. In some cases, it will be possible to bound the degree of approximation, but a run-time monitor can only estimate where the optimal solution falls within this bound.

A similar observation can be made about other classes of problems besides optimization problems. For problems that involve estimating a point value, the difference between the estimated point value and the true point value can't be known until the algorithm has converged to an exact value (Horvitz, Suermondt, & Cooper 1989). For anytime problem-solvers that rely on abstraction to create approximate solutions, solution quality may be difficult to assess for other reasons. For example, it may be difficult for a run-time monitor to predict the extent of planning needed to fill in the details of an abstract plan (Zilberstein 1996). We conclude that for many problems, the best a run-time monitor can do is estimate the quality of an anytime solution with some degree of probability.

Monitoring based on estimated quality

When the quality of approximate solutions cannot be accurately measured at run-time, the success of run-time monitoring requires solving two new problems. First, some reliable method must be found for estimating solution quality at run-time. It is impossible to specify a universal method for this – how solution quality is estimated will vary from algorithm to algorithm. We sketch a general approach and, in the section that follows, describe an illustrative example. The second problem is that a monitoring policy must be conditioned on the estimate of solution quality rather than solution quality itself.

To solve these problems, we condition a run-time estimate of solution quality on some feature f_r of the currently available solution that is correlated with solution quality. When a feature is imperfectly correlated with solution quality, we have also found it useful to condition the estimate on the running time of the algorithm, t_k. Conditioning an estimate of solution quality on the algorithm's running time as well as some feature observed by a run-time monitor provides an important guarantee; it ensures that the estimate will be at least as good as if it were based on running time alone.

As a general notation, let $Pr(q_i | f_r, t_k)$ denote the probability that the currently available solution has quality q_i when the run-time monitor observes feature f_r after running time t_k. In addition, let $Pr(f_r|q_i, t_k)$ denote the probability that the run-time monitor will observe feature f_r if the currently available solution has quality q_i after running time t_k. Again, these probabilities can be determined from statistical analysis of the behavior of the algorithm. These "partial observability" functions, together with the dynamic performance profile defined earlier, can be used to calculate the following probabilities for use in predicting the improvement of solution quality after additional time allocation Δt when the quality of the currently available solution can only be estimated.

$$Pr(q_j|f_r, t_k, \Delta t) = \sum_i Pr(q_i|f_r, t_k) Pr(q_j|q_i, \Delta t)$$
$$Pr(f_s|f_r, t_k, \Delta t) = \sum_i Pr(q_i|f_r, t_k) \sum_j Pr(q_j|q_i, \Delta t) Pr(f_s|q_j, t_k+\Delta t) - C$$

These probabilities can also be determined directly from statistical analysis of the behavior of the algorithm, without the intermediate calculations. In either case, these probabilities make it possible to find a monitoring policy by optimizing the following value function using dynamic programming.

$$V(f_r, t_k) = \max_{\Delta t, m} \begin{bmatrix} \sum_j Pr(q_j|f_r, t_k, \Delta t) U(q_j, t_k+\Delta t) \\ \text{if } m = \text{stop}, \\ \sum_s Pr(f_s|f_r, t_k, \Delta t) V(f_s, t_k+\Delta t) - C \\ \text{if } m = \text{monitor} \end{bmatrix}$$

The resulting policy may not be optimal in the sense that it may not take advantage of all possible run-time evidence about solution quality, for example, the trajectory of observed improvement. Finding an optimal policy may require formalizing the problem as a partially observable Markov decision process and using computationally intensive algorithms developed for such problems (Cassandra, Littman, & Kaelbling 1994). The approach we have described is simple and efficient, however, and it provides an important guarantee: it only recommends monitoring if it results in a higher expected value than allocating a fixed running time without monitoring. This makes it possible to distinguish cases in which monitoring is cost-effective from cases in which it is not. Whether monitoring is cost-effective will depend on the variance of the performance profile, the time-dependent utility of the solution, how well the quality of the currently available solution can be estimated by the run-time monitor, and the cost of monitoring – all factors weighed in computing the monitoring policy.

Example

As an example of how this technique can be used to determine a combined policy for monitoring and stopping, we apply it to a tour improvement algorithm for the traveling salesman problem developed by Lin and Kernighan (1973).

quality	$\frac{\textit{Length}(\textit{Current tour})}{\textit{Length}(\textit{Optimal tour})}$
5	$1.05 \to 1.00$
4	$1.10 \to 1.05$
3	$1.20 \to 1.10$
2	$1.35 \to 1.20$
1	$1.50 \to 1.35$
0	$\infty \to 1.50$

Table 1: Discretization of solution quality.

feature	$\frac{\textit{Length}(\textit{Current tour})}{\textit{Length}(\textit{Lower bound})}$
6	$1.3 \to 1.0$
5	$1.4 \to 1.3$
4	$1.5 \to 1.4$
3	$1.6 \to 1.5$
2	$1.7 \to 1.6$
1	$2.0 \to 1.7$
0	$\infty \to 2.0$

Table 2: Discretization of feature correlated with solution quality.

This local optimization algorithm begins with an initial tour, then repeatedly tries to improve the tour by swapping random paths between cities. The example is representative of anytime algorithms that have variance in solution quality as a function of time.

We defined solution quality as the approximation ratio of a tour,

$$\frac{Length(\text{Current tour})}{Length(\text{Optimal tour})}$$

and discretized this metric using Table 1. The maximum running time of the algorithm was discretized into twelve time-steps, with one time-step corresponding to approximately 0.005 CPU seconds. A dynamic performance profile was compiled by generating and solving a thousand random twelve-city traveling salesman problems. The time-dependent utility of a solution of quality q_i at time t_k was arbitrarily defined by the function

$$U(q_i, t_k) = 100q_i - 20t_k.$$

Note that the first term of the utility function can be regarded as the *intrinsic value* of a solution and the second term as the *time cost*, as defined by Russell and Wefald (1991).

Without monitoring, the optimal running time of the algorithm is eight time-steps, with an expected value of 269.2. Assuming solution quality can be measured accurately by the run-time monitor (an unrealistic assumption in this case) and assuming a monitoring cost of 1, the dynamic programming algorithm described earlier computes the monitoring policy shown in Table 3. The number in each cell of Table 3 represents how much additional time to allocate to the

quality	start	1	2	3	4	5	6	7	8	9	10	11
						time-step						
5		0	0	0	0	0	0	0	0	0	0	0
4		1M	1M	1M	1M	1M	1M	1M	1M	1M	1	0
3		1M	1M	1M	1M	1M	1M	1M	1M	1M	1	0
2		3M	3M	3M	3M	3M	3M	3M	3M	2	1	0
1		4M	4M	4M	4M	4M	5	4	3	2	1	0
0	5M	5M	5M	5M	5M	6	5	4	3	2	1	0

Table 3: Optimal policy based on actual solution quality.

feature	start	1	2	3	4	5	6	7	8	9	10	11
						time-step						
6				0	0	0	0	0	0	0	0	0
5			1	1	1	1	0	0	0	0	0	0
4			2M	2M	1M	1M	1M	1M	1	0	0	0
3		4M	3M	2M	1M	1M	1M	1M	1M	1	0	0
2		4M	3M	2M	2M	2M	2M	1M	3	2	1	0
1		4M	3M	3M	3M	3M	3M	2M	2M	1M	1	
0	5M	4M	3M	3M	3M	3M	3M					

Table 4: Policy when solution quality is estimated.

algorithm based on the observed quality of the solution and the current time. The letter M next to a number indicates a decision to monitor at the end of this time allocation, and possibly allocate additional running time; if no M is present, the decision is to stop at the end of this time allocation without monitoring. The policy has an expected value of **303.3**, better than the expected value of allocating a fixed running time despite the added cost of monitoring. Its improved performance is due to the fact that the run-time monitor can stop the algorithm after anywhere from 5 to 11 time steps, depending on how quickly the algorithm finds a good result. (If there is no cost for monitoring, a policy that monitors every time step has an expected value of 309.4.)

The policy shown in Table 3 was constructed by assuming the actual quality of an approximate solution could be measured by the run-time monitor, an unrealistic assumption because measuring the quality of the current tour requires knowing the length of an optimal tour. The average length of an optimal tour can provide a very rough estimate of the optimal tour length in a particular case, and this can be used to estimate the quality of the current tour. For a traveling salesman problem that satisfies the triangle inequality, however, much better estimates can be made by using one of a number of algorithms for computing a lower bound on the optimal tour length (Reinelt 1994). Computing a lower bound involves solving a relaxation of the problem; it is analogous to an admissable heuristic function in search. For a traveling salesman problem that satisfies the triangle inequality, there exist polynomial-time algorithms that can compute a lower bound that is on average within two or three percent of the optimal tour length. For our test, however, we used Prim's minimal spanning tree algorithm that very quickly computes a bound that is less tight, but still correlated with the optimal tour length. The feature

$$\frac{Length(\text{Current tour})}{Length(\text{Lower bound})}$$

was discretized using Table 2. The cost overhead of monitoring consists of computing the lower bound at the beginning of the algorithm and monitoring the current tour length at intervals thereafter.

Table 4 shows the monitoring policy given a monitoring cost of 1, when an estimate of solution quality is conditioned on both this feature and the running time of the algorithm. The expected value of the policy is **282.3**, higher than for allocating a fixed running time without monitoring but lower than if the run-time monitor could determine the actual quality of an approximate solution. As this demonstrates, the less accurately a run-time monitor can measure the quality of an approximate solution, the less valuable it is to monitor.

When an estimate of solution quality is based only on this feature, and not also on running time, the expected value of monitoring is **277.0**. This is still an improvement over not monitoring, but the performance is not as good as when an estimate is conditioned on running time as well. Because conditioning a dynamic performance profile on running time significantly increases its size, however, this tradeoff may be acceptable in cases when the feature used to estimate quality is very reliable. For all of these results, the improved performance predicted by dynamic programming was confirmed by simulation experiments.

For the tour improvement algorithm, variance in solution quality over time is minor and the improved performance with run-time monitoring correspondingly small. We plan to apply this technique to other problems for which variance in solution quality is larger and the payoff for run-time monitoring promises to be more significant. However, the fact that this technique improves performance even when variance is small, solution quality is difficult to estimate at run-time, and monitoring incurs a cost, supports its validity and potential value.

Conclusion

The framework developed in this paper extends previous work on meta-level control of anytime algorithms. One contribution is the use of dynamic programming to compute a non-myopic stopping rule. Previous schemes for run-time monitoring have relied on myopic computation of the expected value of continued deliberation to determine a stopping time (Breese & Horvitz 1991; Horvitz 1990), although Horvitz has also recommended various degrees of lookahead search to overcome the limitations of a myopic approach. Because dynamic programming is particularly well-suited for off-line computation of a stopping rule, it is also an example of what Horvitz calls compilation of metareasoning.

Another contribution of this framework is that it makes it possible to find an intermediate strategy between continuous monitoring and not monitoring at all. It can recognize whether or not monitoring is cost-effective, and when it is, it can adjust the frequency of monitoring to optimize utility. An interesting property of the monitoring policies found is that they recommend monitoring more frequently near the expected stopping time of an algorithm, an intuitive strategy.

Perhaps the most significant aspect of this framework is that it makes it possible to evaluate tradeoffs between various factors that influence the utility of monitoring. For example, the dynamic programming technique is sensitive to both the cost of monitoring and to how well the quality of the currently available solution can be estimated by the run-time monitor. This makes it possible to evaluate a tradeoff between these two factors. Most likely, there will be more than one method for estimating a solution's quality and the estimate that takes longer to compute will be more accurate. Is the greater accuracy worth the added time cost? The framework developed in this paper can be used to answer this question by computing a monitoring policy for each method and comparing their expected values to select the best one.

Acknowledgments

Support for this work was provided in part by the National Science Foundation under grant IRI-9409827 and in part by Rome Laboratory, USAF, under grant F30602-95-1-0012.

References

Bertsekas, D.P. 1987. *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs, N.J.: Prentice-Hall.

Boddy, M., and Dean., T. 1994. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence* 67:245-285.

Breese, J.S., and Horvitz, E.J. 1991. Ideal reformulation of belief networks. Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, 129-143.

Cassandra, A.R.; Littman, M.L.; and Kaelbling, L.P. 1994. Acting optimally in partially observable stochastic domains. Proceedings of the Twelth National Conference on Artificial Intelligence, 1023-1028.

Horvitz, E.J.; Suermondt, H.J.; and Cooper, G.F. 1989. Bounded conditioning: Flexible inference for decisions under scarce resources. Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence.

Horvitz, E.J. 1990. Computation and Action under Bounded Resources. PhD Thesis, Stanford University.

Lin, S., and Kernighan, B.W. 1973. An effective heuristic algorithm for the Traveling Salesman problem. *Operations Research* 21:498-516.

Paul, C.J.; Acharya, A.; Black, B.; and Strosnider, J.K. 1991. Reducing problem-solving variance to improve predictability. *CACM* 34(8):80-93.

Reinelt, G. 1994. *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag.

Russell, S., and Wefald, E. 1991. *Do the Right Thing: Studies in Limited Rationality*. The MIT Press.

Shekhar, S., and Dutta, S. 1989. Minimizing response times in real time planning and search. Proceedings of the Eleventh IJCAI, 238-242.

Smith, K.P., and Liu, J.W.S. 1989. Monotonically improving approximate answers to relational algebra queries. *COMPSAC-89*, Orlando, Florida.

Wellman, M.P., and Liu, C.-L. 1994. State-space abstraction for anytime evaluation of probabilistic networks. Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, 567-574.

Zilberstein, S. 1993. Operational Rationality through Compilation of Anytime Algorithms. Ph.D. dissertation, Computer Science Division, University of California at Berkeley.

Zilberstein, S. 1996. Resource-bounded sensing and planning in autonomous systems. To appear in *Autonomous Robots*.

Zilberstein S., and Russell S. 1995. Approximate reasoning using anytime algorithms. In S. Natarajan (Ed.), *Imprecise and Approximate Computation*, Kluwer Academic Publishers.