Using Performance Profile Trees to Improve Deliberation Control

Kate Larson and **Tuomas Sandholm**
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
{klarson,sandholm}@cs.cmu.edu

Abstract

Performance profile trees have recently been proposed as a theoretical basis for fully normative deliberation control. In this paper we conduct the first experimental study of their feasibility and accuracy in making stopping decisions for anytime algorithms on optimization problems. Using data and algorithms from two different real-world domains, we compare performance profile trees to other well-established deliberation-control techniques. We show that performance profile trees are feasible in practice and lead to significantly better deliberation control decisions. We then conduct experiments using performance profile trees where deliberation-control decisions are made using conditioning on multiple features of the solution to illustrate that such an approach is feasible in practice.

Introduction

In many AI applications, bounded rationality is simply a necessary evil that has to be dealt with. The realities of limited computational resources and time pressures caused by real-time environments mean that agents are not always able to optimally determine their best decisions and actions. The field of artificial intelligence has long searched for useful techniques for coping with this problem. Herbert Simon advocated that agents should forgo perfect rationality in favor of limited, economical reasoning. His thesis was that "the global optimization problem is to find the least–cost, or best–return decision, net of computational costs" (Simon 1982).

Considerable work has focused on developing *normative* models that prescribe how a computationally limited agent *should* behave (see, for example (Horvitz 2001; Russell 1997)). In particular, decision theory has proved to be a powerful tool in developing approaches to meta-level control of computation (Horvitz 1990). In general this is a highly nontrivial undertaking, encompassing numerous fundamental and technical difficulties. As a result, most of these methods resort to simplifying assumptions such as myopic search control (Russell and Wefald 1991; Baum and Smith 1997), assuming that an algorithm's future performance can be deterministically predicted using a performance profile (Horvitz 1988; Boddy and Dean 1994), assuming that an algorithm's future performance does not depend on the run on that instance so far (Zilberstein and Russell 1996; Zilberstein *et al.* 1999; Horvitz 2001), resorting to asymptotic notions of bounded optimality (Russell and Subramanian 1995), or using myopic approaches for decision making (Breese and Horvitz 1990). While such simplifications can be acceptable in single-agent settings as long as the agent performs reasonably well, any deviation from full normativity can be catastrophic in multiagent systems. If a multiagent system designer can not guarantee that a strategy (including deliberation actions) is the best strategy that an agent can use, there is a risk that the agent is motivated to use some other strategy. Even if that strategy happens to be "close" to the desired one, the social outcome may be far from desirable. Therefore, a fully normative deliberation control method, which considers all possible information and ways an agent can make a deliberation decision, is required as a basis for analyzing the strategies of agents.

Recently, a fully normative deliberation control method, the *performance profile tree*, was introduced for making stopping decisions for any anytime optimizer (treated as a black box) (Larson and Sandholm 2001). This deliberation control method has been used to analyze (deliberation) strategies of agents in different bargaining and auction settings in order to understand the repercussions that limited deliberation resources have on agents' game-theoretic behavior. However, a weakness of this work is that it has been highly theoretical. While the full normativity provided by the performance profile tree is required theoretically, it has been unclear whether the performance profile tree is of practical use.

In this paper we experimentally study the use of performance profile trees to determine their practicality and usefulness for helping a single agent decide when to stop its anytime optimization algorithm. On data generated from black-box anytime problem solvers, we illustrate that it is feasible to use performance profile tree based deliberation control in hard real-world problems. We also show that this leads to more accurate deliberation control decisions than the use of the performance profile representations presented in prior literature. Furthermore, we demonstrate that the performance profile tree can easily handle conditioning its decisions on (the path of) other solution features in addition to solution quality.

The paper is organized as follows. In the next section we provide an overview of deliberation control along with details of the different methods used in the experiments. After this we provide a description of the setup of the experiments. We describe the problem domains which we use and explain how the performance profiles are created. This is followed by the presentation and discussion of the results, after which we conclude.

Decision-theoretic deliberation control

We begin by providing a short overview of deliberation control methods, and then describe in detail the different methods that are tested in this paper. We assume that agents have algorithms that allow them to trade off computational resources for solution quality. In particular, we assume that agents have *anytime algorithms*, that is, algorithms that improve the solution over time and return the best solution available even if not allowed to run to completion (Horvitz 1988; Boddy and Dean 1994). Most iterative improvement algorithms and many tree search algorithms (such as branch and bound) are anytime algorithms, additionally there has been many successful applications of anytime algorithms in areas like belief and influence diagram evaluation (Horsch and Poole 1999), planning and scheduling (Boddy and Dean 1994), and information gathering (Grass and Zilberstein 2000).

If agents have infinite computing resources (i.e. no deadlines and free computation), they would be able to compute the optimal solutions to their problems. Instead, in many settings agents have *time-dependent* utility functions. That is, the utility of an agent depends on both the solution quality obtained, and the amount of time spent getting it,

$$U(q,t) = u(q) - \mathrm{cost}(t)$$

where $u(q)$ is the utility to the agent of getting a solution with quality q and $\mathrm{cost}(t)$ is the cost incurred of computing for t time steps.

While anytime algorithms are models that allow for the trading off of computational resources, they do not provide a complete solution. Instead, anytime algorithms need to be paired with a meta-level deliberation controller that determines how long to run the anytime algorithm, that is, when to stop deliberating and act with the solution obtained. The deliberation controller's stopping policy is based on a *performance profile*: statistical information about the anytime algorithm's performance on prior problem instances. This helps the deliberation controller project how much (and how quickly) the solution quality would improve if further computation were allowed.[1] Performance profiles are usually generated by prior runs of the anytime algorithm on different problem instances.

There are different ways of representing performance profiles. At a high level, performance profiles can be classified as being either *static* or *dynamic*.

[1] In the rest of this paper, when it is obvious from the context, we will refer to the combination of the deliberation controller and the performance profile as just the performance profile.

Static performance profiles predict, before the anytime algorithm starts running on the instance at hand, the optimal stopping time for the algorithm. The *performance profile curve (PPCurve)* (Horvitz 1988; Boddy and Dean 1994), Figure 1(a), is an example of a static performance profile. It is created by averaging, at each time point, the solution quality obtained on prior runs (on different instances). Given the curve, specified as solution quality as a function of time, $q(t)$, as well as the cost function $\mathrm{cost}(t)$, the deliberation policy of an agent is to allocate time t^* to the algorithm, where

$$t^* = \arg\max_t \left[u(q(t)) - \mathrm{cost}(t)\right].$$

A potential weakness with static performance profiles is that the time allocation decision is made before the algorithm starts. Instead, by monitoring the current run of the algorithm, it may be possible to gather additional information which leads to better predictions about future changes in solution quality, and thus enables better stopping decisions.

Dynamic performance profiles monitor the run of the algorithm and make online decisions about whether to allow the algorithm more time. Dynamic performance profiles are often represented as a table of discrete values which specify a discrete probability distribution over solution quality for each time step (Zilberstein and Russell 1996), Figure 1(b). The basic table-based representation (Zilberstein and Russell 1996) loses some information that can be useful for making stopping decisions. In particular, it cannot condition the projection on the current solution. Consider Figure 1(b). Once the shaded cell is reached, it is clear that there should be no probability mass at cells of lower solution quality at future steps, but the representation cannot capture that.

To address this, Hansen and Zilberstein (Hansen and Zilberstein 2001) proposed a dynamic programming approach where the projection of solution quality is conditioned on the current solution quality. In particular, by defining the utility of having a solution of quality q_i at time t_k to be $U(q_i,t_k) = u(q_i) - \mathrm{cost}(t_k)$, a stopping rule can be found by optimizing the following value function

$$V(q_i,t_k) = \max_d \begin{cases} U(q_i,t) & \text{if } d = \text{stop} \\ \sum_j P(q_j|q_i,\Delta t)V(q_j,t+\Delta t) & \text{if } d = \text{continue} \end{cases}$$

to determine the policy

$$\pi(q_i,t) = \mathrm{argmax}_d \begin{cases} U(q_i,t) & \text{if } d = \text{stop} \\ \sum_j P(q_j|q_i,\Delta t)V(q_j,t+\Delta t) & \text{if } d = \text{continue} \end{cases}$$

Throughout the rest of this paper, when we compare our methods against the table-based approach, we always use the dynamic program in conjunction with the table in order to give the table-based approach the best possible chance. For short, we will call that combination a *performance profile table (PPTable)*.

Even the PPTable approach yields suboptimal stopping decisions because the representation loses information. In particular, all information about the path of the run is lost. In Figure 1(b), the shaded cell could have been reached by path A or path B. The solution quality projection should be

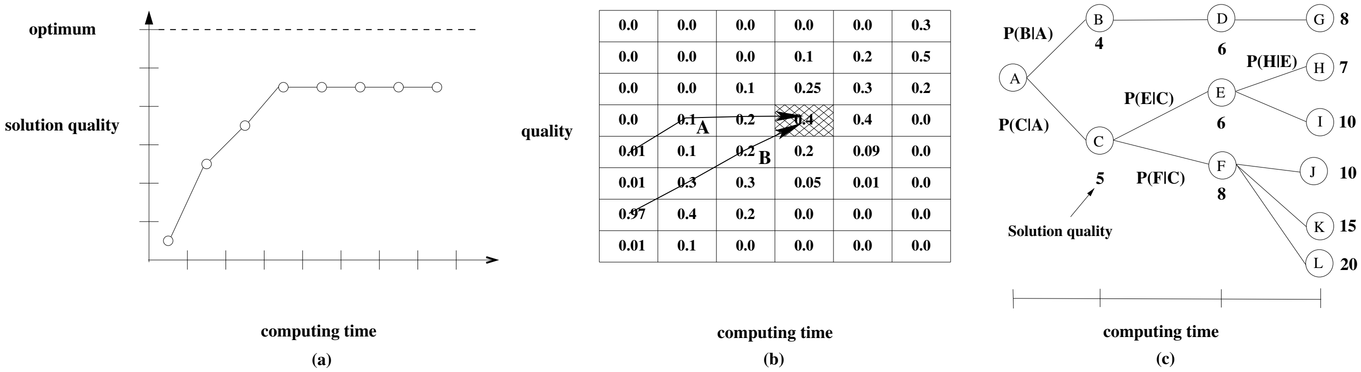

Figure 1: *Three performance profile representations: a) performance profile curve (PPCurve), b) performance profile table (PPTable), and c) a performance profile tree (PPTree).*

more optimistic if the cell was reached via path B, but using the PPTable, the same stopping decision must be made in both cases. (However, the PPTable yields optimal stopping decisions in settings where the current solution quality captures all of the information from the path that is pertinent for projecting solution quality. This Markov assumption is generally not satisfied, as the example above shows.)

In order to capture all of the information available for making stopping decisions, we introduced the *performance profile tree* (PPTree) representation (Larson and Sandholm 2001). In a PPTree, the nodes represent solutions at given time points, while each edge carries the probability that the child node is reached given that the parent was reached. Figure 1(c) exemplifies one such tree. A PPTree can support conditioning on any and all features that are deemed to be of importance for making stopping decisions since the nodes can hold information about solution quality and any other solution feature that may be important. For example, in scheduling applications, often the *slack* in a schedule is a good predictor of future improvement. The solution information stored at a node could therefore include a variable for solution quality and a variable for slack.

A key aspect of the PPTree is that it automatically supports conditioning on the *path* so far. The performance profile tree that applies given a path of computation is the subtree rooted at the current node n. This subtree is denoted by $\mathcal{T}(n)$. If an agent is at a node n with solution quality $q(n)$, then when estimating how much additional computing would increase the solution quality, the agent need only consider paths that emanate from node n. The probability, $P_n(n')$, of reaching a particular future node n' in $\mathcal{T}(n)$ is simply the product of the probabilities on the path from n to n'. The expected solution quality after allocating t more time steps to the problem, if the current node is n, is $\sum P_n(n') \cdot q(n')$ where the sum is over the set $\{n' | n'$ is a node in $\mathcal{T}_i(n)$ which is reachable in t time steps$\}$.

Deliberation policies for the PPTree are constructed by optimizing the following rule

$$V(q(n),t) = \max_d \begin{cases} U(q(n),t) & \text{if d=stop} \\ \sum_{n'} P_n(n') V(q(n'), t+\Delta t) & \text{if d=continue} \end{cases}$$

where $n' \in \{\text{nodes in } \mathcal{T}(n) \text{ at depth } \Delta t \text{ time steps}\}$, to determine the policy

$$\pi(q(n),t) = \text{argmax}_d \begin{cases} U(q(n),t) & \text{if d=stop} \\ \sum_{n'} P_n(n') V(q(n'), t+\Delta t) & \text{if d=continue} \end{cases}$$

Experimental setup

While a fully normative deliberation control method is required for game-theoretic analysis of computationally limited agents, to date no experimental work has been done to show that (1) the performance profile tree based deliberation control method is feasible in practice, and (2) that in practice such sophisticated deliberation control is any better than earlier decision-theoretic deliberation control methods that relied on simpler performance profile representations . In this paper we bridge this gap and show that performance profile trees are desirable in practice also for single-agent deliberation control. In the first set of experiments, we demonstrate (1) and (2). In that experiment we use solution quality as the only feature stored in a tree node (as we mentioned above, the PPTree automatically conditions on the *path* followed to reach the node). In the second set of experiments, we show that it is feasible to use additional problem features to make deliberation decisions.

Our deliberation control method is domain independent and domain problem solver independent—yielding a clean separation between the domain problem solver (a black box) and the deliberation controller. This separation allows one to develop deliberation control methodology that can be leveraged across applications. To demonstrate this we conduct experiments in two different application domains using software which was developed independently from the deliberation controllers.

Example domain problem solving environments

We conducted our experiments in two different domain environments – vehicle routing and single-machine manufacturing scheduling.

Vehicle routing In the real-world vehicle routing problem (VRP) in question, a dispatch center is responsible for a certain set of tasks (deliveries) and has a certain set of resources (trucks) to take care of them (Sandholm 1993; Sandholm and Lesser 1997). Each truck has a depot, and each delivery has a pickup location and a drop-off location. The dispatch center's problem is to minimize transportation cost (driven distance) while still making all of its deliveries and honoring the following constraints: 1) each vehicle has to begin and end its tour at its depot, and 2) each vehicle has a maximum load weight and maximum load volume constraint. This problem is $\mathcal{NP}$-complete.

To generate data for our experiments, an iterative improvement algorithm was used for solving the VRP. The center initially assigned deliveries to trucks in round-robin order. The algorithm then iteratively improved the solution by selecting a delivery at random, removing it from the solution, and then reinserting it into the least expensive place in the solution (potentially to a different truck, and with pickup potentially added into a different leg of the truck's route than the drop-off) without violating any of the constraints. Each addition-removal is considered one iteration. We let the algorithm run until there was no improvement in the solution for some predefined number, k, of steps. Figure 2 shows the results of several runs of this iterative improvement algorithm on different instances used in the experiments, with $k = 250$. The algorithm clearly displays diminishing returns to scale, as is expected from anytime algorithms.

The problem instances were generated using real-world data collected from a dispatch center that was responsible for 15 trucks and 300 deliveries. We generated training and testing sets by randomly dividing the deliveries into a set of 210 training deliveries and 90 testing deliveries. To generate a training (testing) instance, we randomly selected (with replacement) 60 deliveries from the training (testing) set.

Manufacturing scheduling The second domain is a single-machine manufacturing scheduling problem with sequence-dependent setup times on the machines, where the agent's objective is to minimize weighted tardiness $= \sum_{j \in J} w_j T_j = \sum_{j \in J} w_j \max(f_j - d_j, 0)$, where T_j is the tardiness of job j, and w_j, f_j, d_j are the weight, finish time, and due-date of job j.

In our experiments, we used a state-of-the-art scheduler developed by others (Cicirello and Smith 2002) as the domain problem solver. It is an iterative improvement algorithm that uses a scheduling algorithm called Heuristic Biased Stochastic Sampling (Bresina 1996). We treated the domain problem solver as a black box without any modifications.

The problem instances were generated according to a standard benchmark (Lee *et al.* 1997). The due-date tightness factor was set to 0.3 and the due-date range factor was set to 0.25. The setup time severity was set to 0.25. These parameter values are the ones used in standard benchmarks (Lee *et al.* 1997). Each instance consisted of 100 jobs to be scheduled. We generated the training instances and test instances using different random number seeds.

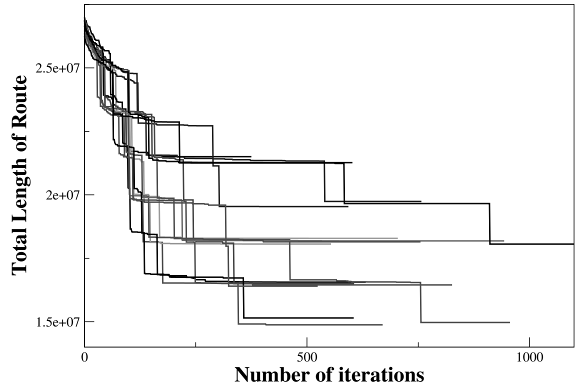

Figure 2: *Runs of the vehicle routing algorithm on different problem instances. The x-axis is number of iterations and the y-axis is the total distance traveled by all trucks.*

Constructing performance profiles

Performance profiles encapsulate statistical information about how the domain problem solver has performed on past problem instances. To build performance profiles, we generated

- 1000 instances for the vehicle routing domain. We ran the algorithm on each instance until there was no improvement in solution quality for 250 iterations.
- 10000 instances for the scheduling domain. We ran the algorithm until there was no improvement in solution quality for 400 iterations.

From this data, we generated the performance profiles using each of the three representations: PPCurve, PPTable, and PPTree. Like PPTable-based deliberation control, PPTree requires discretization of computation time and of solution quality (otherwise no two runs would generally populate the same part of the table/tree, in which case no useful statistical inferences could be drawn).[2]

Computation time was discretized the same way for each of the three performance profile representations. We did this the obvious way in that one iteration of each algorithm (routing and scheduling) was equal to one computation step. For the solution quality we experimented with different discretizations. Due to space limitations we present only results where the scheduling data was discretized into buckets of width 100, and the vehicle routing data was discretized into buckets of width 50000.[3] For the vehicle routing domain there turned out to be 1943 time steps, and 551 buckets of solution quality. For the scheduling domain there turned out to be 465 time steps, and 750 buckets of solution quality.

At first glance it may seem that this implies performance profile trees of size 551^{1943} for the trucking domain and 750^{465} for the scheduling domain. However, most of the paths of the tree were not populated by any run (because

[2] The PPCurve does not require any discretization on solution quality, so we gave it the advantage of no discretization.

[3] The results obtained from these discretizations were representative of the results obtained across all the tested discretizations.

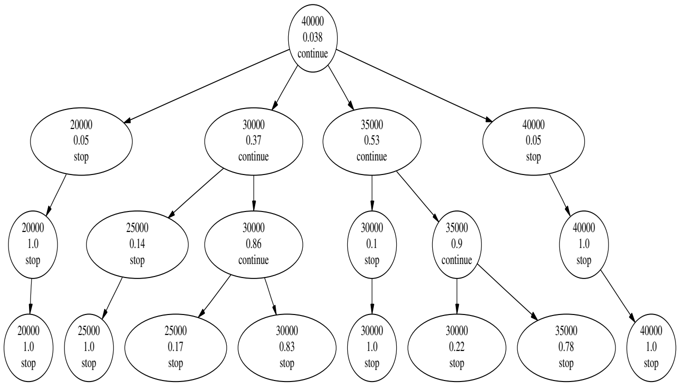

Figure 3: *Subtree of a performance profile tree generated from instances from the scheduling domain. In each node there are three entries. The first entry is the (discretized) solution quality. The second entry is the probability of reaching the node, given that its parent was reached. The final entry represents the stopping policy. An entry labelled "continue" means that the agent should compute another step, while an entry labelled "stop" means that the agent should stop all computation and act with the solution at hand.*

there are "only" 1000 (or 10000) runs, one per instance). We generated the tree dynamically in memory, such that only the populated parts were stored. This way, we could be assured that the number of edges in the tree for trucking was at most 1943 × #instances (because each instance can generate at most one new edge for each compute step). Similarly, for scheduling it was 465 × #instances.

In practice, the trees were easy to generate, but they are much too large to lay out in a paper. Therefore, in Figure 3 we present a subtree of a performance profile tree generated from the same 10000 scheduling instances, but with a much coarser discretization: buckets of width 5000 on solution quality and computing steps that include 150 iterations each (yielding 4 buckets on the computation time axis, i.e., 4 depths to the tree—with root at depth 0).

Cost functions

In all the experiments we used cost functions of the form $C \cdot t$ where C was an exogenously given cost of one step of computing and t was the number of time steps computed. We studied the behavior of deliberation control methods under a wide range of values of C. For the vehicle routing domain we used $C \in \{0, 10, 100, 500, 1000, 5000, 10000, 25000, 35000, 50000, 100000, 1000000\}$ In the scheduling domain we used $C \in \{0, 1, 10, 50, 100, 500, 1000, 5000, 10000, 100000\}$. These value choices span the interesting range: at the bottom end, no controller ever stops the deliberation, and at the top end, each controller stops it immediately.

Comparison of performance profiles

In the first set of experiments we tested the feasibility of PPTree-based deliberation control and compared its decision-making effectiveness against other performance-profile representations (PPCurve and PPTable).

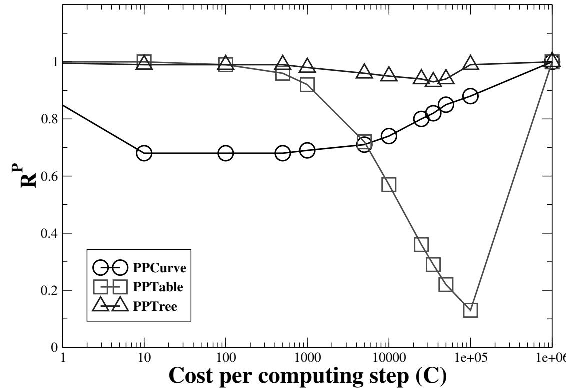

Figure 4: *Performance of the different performance profiles in the vehicle routing domain. Values closer to 1.0 (optimal) are better. PPTree outperforms both PPCurve and PPTable.*

To evaluate the performance, we generated $N = 500$ new test instances of the trucking problem and $N = 5000$ new test instances from the scheduling domain. Each of the three performance profile representations was evaluated on the same test set.

For each test instance we determined the optimal stopping point, t_i^{opt}, given that the entire path of the run was known in hindsight. (This stopping point is optimistic in the sense that it does not use discretization of solution quality. Furthermore, real deliberation controllers do not have hindsight at their disposal.) This allowed us to determine the optimal value that the agent could have possibly gotten on instance i in hindsight: $U_{\mathrm{opt}}(i) = q_i(t_i^{\mathrm{opt}}) + C \cdot t_i^{\mathrm{opt}}$ where $q_i(t)$ is the solution quality after computing for t time steps, and C is the exogenously given cost of one step of computing.[4]

We evaluated the three performance profile representations $P \in \{\mathrm{PPTree}, \mathrm{PPTable}, \mathrm{PPCurve}\}$ separately on the test instances, recording for each instance the stopping point t_i^{P} that deliberation controller, P, imposed and the resulting value. That is, we stored $U_{\mathrm{P}}(i) = q_i(t_i^{\mathrm{P}}) + C \cdot t_i^{\mathrm{P}}$. We determined how far from optimal the resulting value was as a ratio $R_i^{\mathrm{P}} = \frac{U_{\mathrm{opt}}(i)}{U_{\mathrm{P}}(i)}$. Then, $R^{\mathrm{P}} = \frac{1}{N} \sum_i^N R_i^{\mathrm{P}}$ gave an overall measure of performance (the closer R is to 1.0, the better). Figures 4 and 5 display the results.

When computation was free or exorbitantly expensive compared to the gains available from computing, then all the deliberation control methods performed (almost) optimally. With free computing, the deliberation control problem is trivial: it is simply best to compute forever. Similarly, when computation is extremely expensive, it is best to not compute at all. For the midrange costs (i.e., the interesting range), the deliberation controllers performed quite differently from each other. The PPTree outperformed both the PPCurve and the PPTable. In the vehicle routing experiments, the PPTree was, at worst, 93.0% of optimal (when $C = 35000$)

[4]In both application domain, lower solution quality is better. Therefore, we add the cost function, instead of subtracting it in the utility equation.

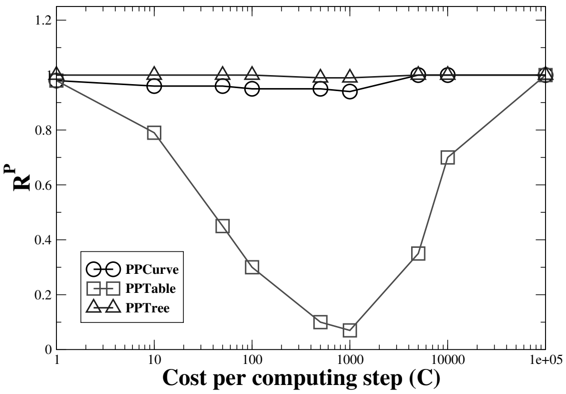

Figure 5: *Performance of the different performance profiles in the scheduling domain. Values closer to 1.0 are better. PPTree out performs both PPTable and PPCurve.*

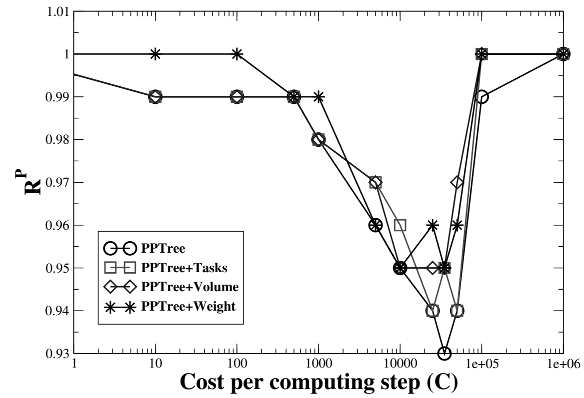

Figure 6: *Performance of the PPTree when deliberation-control decisions are conditioned on both solution quality and an additional solution feature (variance across trucks in number of tasks, variance across trucks of the average volume to max-volume ratio, and variance across trucks in the average weight to max-weight ratio.*

and often performed better (Figure 4). In the scheduling experiments, the PPTree was always within 99.0% of optimal (Figure 5).

In the scheduling domain, the PPCurve performed reasonably well with R^P ranging from 0.95 to 1.00. In the vehicle routing domain, its performance was not as good, with R^P ranging from 0.68 to 1.00. A possible explanation for the difference is that in scheduling there was less variance (in the stopping time) among instances. Therefore, in the scheduling domain the optimal stopping point for the average algorithm run was a better estimate for any given run, compared to the routing domain.

The PPTable had the widest variability in behavior. For both low and high costs it performed well in both application domains. However, for midrange costs it performed poorly, as low as 0.07 in scheduling and 0.13 in vehicle routing. In particular, the PPTable appeared to be overly optimistic when determining a deliberation control policy, as it often allowed too much computing. It was not able to differentiate between algorithm runs which had flattened out and ones in which additional improvement in solution quality was possible. While the dynamic programming approach used in the PPTable produces an optimal policy if solution quality improvement satisfies the Markov property as we discussed earlier, it appears to not be robust when in practice that property does not hold.

While the PPTree performs better than PPCurve and PPTable with respect to absolute difference in performance, it is also important to determine whether there is statistical significance to the results. Using a Fisher sign test, we compared the PPTree against both the PPCurve and the PPTable. We found in both the trucking and scheduling domains that the dominance of the PPTree was truly significant, resulting in $p-$values less than 10^{-13} for all costs where there was any difference in the performance between the performance profiles.[5]

[5] When costs were very low ($C = 0$) or very high ($C = 10^6$ in the trucking domain and $C = 10^5$ in the scheduling domain), all performance profiles performed optimally, and thus, identically.

We also experimented using different solution quality and time step discretizations. The general patterns seen in Figures 4 and 5 were again observed. Furthermore, we ran experiments where smaller training sets (of as low as 50 instances) were used. While the performance of all of the deliberation controllers was adversely affected, the relative ranking of their performance did not change.

In summary, this first set of experiments showed that PPTree-based deliberation control is feasible in practice and outperforms the earlier performance profile representations. It also showed that the method is close to optimal even when solution quality and computation time are discretized and conditioning is done only on the path of the solution quality (instead of additional/all possible predictive features).

Conditioning on (the path of) additional solution features

Using the PPTree as our performance profile representation, we experimented using additional solution features to help in making deliberation control decisions. In the vehicle routing domain, in addition to solution quality (total distance driven) we also allowed conditioning on the following features: 1) variance across trucks in the number of tasks, 2) variance across trucks in the truck's average weight to max-weight ratio, and 3) variance across trucks in the truck's average volume to max-volume ratio. The results are reported in Figure 6. Using the additional solution features did lead to slight improvement in the quality of stopping decisions (a 3.0% absolute improvement for $C = 25000$; less improvement for other values of C). Similarly, in experiments conducted in the scheduling domain where slackness in the schedule was used as an additional feature, there was also slight improvement in the deliberation control results.

While the gains from using additional problem instance features in our experiments seem small, it must be recalled that the accuracy from conditioning on only solution quality was close to optimal already. For example when $C = 25000$ the overall performance improved from 94.0% to 97.0% of optimal. So we are starting to hit a ceiling effect. Using a Fisher sign test to analyze the significance of improvement in solution quality when using additional features, we found that for all features there was little significant improvement. Again, we hypothesize that a ceiling effect is in action. In summary, the second set of experiments demonstrated that performance profile trees can support conditioning on (the path of) multiple solution features in practice.

Conclusions and future research

Performance profile tree based stopping control of anytime optimization algorithms has recently been proposed as a theoretical basis for fully normative deliberation control. In this paper, we compared different performance profile representations in practice, and showed that the performance profile tree is not only feasible to construct and use, but also leads to better deliberation control decisions than prior methods. We then conducted experiments were deliberation control decisions were conditioned on (the path of) multiple solution features (not just solution quality), demonstrating feasibility of that idea.

There are several research directions that can be taken at this point. First, the performance profile tree was developed as a model of deliberation control for multiagent systems. It would be interesting to conduct a series of experiments to see the efficacy of the approach in multiagent settings. Second, we believe that more work on identifying solution features which help in deliberation control would be a beneficial research endeavor.

Acknowledgements

We would like to thank Vincent Cicirello and Stephen Smith for providing us with both the scheduler and the scheduling instance generator. The material in this paper is based upon work supported by the National Science Foundation under CAREER Award IRI-9703122, Grant IIS-9800994, ITR IIS-0081246, and ITR IIS-0121678.

References

[Baum and Smith 1997] Eric B Baum and Warren D Smith. A Bayesian approach to relevance in game playing. *Artificial Intelligence*, 97(1–2):195–242, 1997.

[Boddy and Dean 1994] Mark Boddy and Thomas Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67:245–285, 1994.

[Breese and Horvitz 1990] John Breese and Eric Horvitz. Ideal reformulation of belief networks. In *Proceedings of UAI'90*, pages 129–144, 1990.

[Bresina 1996] J. L. Bresina. Heuristic-biased stochastic search. In *Proceedings of AAAI'96*, pages 271–278, 1996.

[Cicirello and Smith 2002] Vincent Cicirello and Steven Smith. Amplification of search performance through randomization of heuristics. In *Principles and Practice of Constraint Programming – CP 2002*, pages 124–138, 2002.

[Grass and Zilberstein 2000] Joshua Grass and Shlomo Zilberstein. A value-driven system for autonomous information gathering. *Journal of Intelligent Information Systems*, 14(1):5–27, 2000.

[Hansen and Zilberstein 2001] Eric Hansen and Shlomo Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126:139–157, 2001.

[Horsch and Poole 1999] Michael Horsch and David Poole. Estimating the value of computation in flexible information refinement. In *Proceedings of UAI'99*, pages 297–304, 1999.

[Horvitz 1988] Eric J. Horvitz. Reasoning under varying and uncertain resource constraints. In *Proceedings of AAAI'88*, pages 111–116, St. Paul, MN, August 1988.

[Horvitz 1990] Eric Horvitz. *Computation and actions under bounded resources*. PhD thesis, Stanford University, 1990.

[Horvitz 2001] Eric Horvitz. Principles and applications of continual computation. *Artificial Intelligence*, 126:159–196, 2001.

[Larson and Sandholm 2001] Kate Larson and Tuomas Sandholm. Bargaining with limited computation: Deliberation equilibrium. *Artificial Intelligence*, 132(2):183–217, 2001.

[Lee *et al.* 1997] Y. H. Lee, K. Bhaskaran, and M. Pinedo. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, 29:45–52, 1997.

[Russell and Subramanian 1995] Stuart Russell and Devika Subramanian. Provably bounded-optimal agents. *Journal of Artificial Intelligence Research*, 1:1–36, 1995.

[Russell and Wefald 1991] Stuart Russell and Eric Wefald. *Do the right thing: Studies in Limited Rationality*. The MIT Press, 1991.

[Russell 1997] Stuart Russell. Rationality and intelligence. *Artificial Intelligence*, 94(1):57–77, 1997.

[Sandholm and Lesser 1997] Tuomas Sandholm and Victor Lesser. Coalitions among computationally bounded agents. *Artificial Intelligence*, 94(1):99–137, 1997.

[Sandholm 1993] Tuomas Sandholm. An implementation of the contract net protocol based on marginal cost calculations. In *Proceedings of AAAI'93*, pages 256–262, Washington, D.C., July 1993.

[Simon 1982] Herbert A Simon. *Models of bounded rationality*, volume 2. MIT Press, 1982.

[Zilberstein and Russell 1996] Shlomo Zilberstein and Stuart Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1–2):181–213, 1996.

[Zilberstein *et al.* 1999] Shlomo Zilberstein, François Charpillet, and Philippe Chassaing. Real-time problem solving with contract algorithms. In *Proceedings of IJCAI'99*, pages 1008–1013, Stockholm, Sweden, 1999.