Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Meta-Level Control of Anytime Algorithms with Online Performance Prediction

Justin Svegliato and **Kyle Hollins Wray** and **Shlomo Zilberstein**
College of Information and Computer Sciences
University of Massachusetts Amherst
{jsvegliato,wray,shlomo}@cs.umass.edu

Abstract

Anytime algorithms enable intelligent systems to trade computation time with solution quality. To exploit this crucial ability in real-time decision-making, the system must decide when to interrupt the anytime algorithm and act on the current solution. Existing meta-level control techniques, however, address this problem by relying on significant offline work that diminishes their practical utility and accuracy. We formally introduce an online performance prediction framework that enables meta-level control to adapt to each instance of a problem without any preprocessing. Using this framework, we then present a meta-level control technique and two stopping conditions. Finally, we show that our approach outperforms existing techniques that require substantial offline work. The result is efficient nonmyopic meta-level control that reduces the overhead and increases the benefits of using anytime algorithms in intelligent systems.

1 Introduction

Anytime algorithms have been developed for a wide range of real-time planning and decision-making tasks, such as belief-space planning [Pineau *et al.*, 2003], probabilistic inference [Ramos and Cozman, 2005], heuristic search [Hansen *et al.*, 1997; Richter *et al.*, 2010], motion planning [Luna *et al.*, 2013], and object detection [Richtsfeld *et al.*, 2013]. Simply put, an anytime algorithm is an algorithm that gradually improves the quality of a solution as it runs and returns the current solution if it is interrupted. This behavior enables an anytime algorithm to trade computation time with solution quality. In intelligent systems, this has proven to be useful since they must often approximate solutions to complex problems in order to respond within an acceptable amount of time. However, to exploit the trade-off between solution quality and computation time, the system must solve a non-trivial meta-level control problem: it must decide when to interrupt the anytime algorithm and act on the current solution.

There have been substantial efforts to develop effective meta-level control for anytime algorithms. One approach estimates the stopping point of the algorithm before it begins and lets it run until that stopping point has been reached [Horvitz, 1987; Boddy and Dean, 1994]. Since the stopping point is not revised after the algorithm begins, this approach is called *fixed allocation*. If there is little uncertainty about the performance of the algorithm or the urgency for the solution, fixed allocation interrupts the algorithm near the optimal stopping point. However, in real-time decision-making problems, there is often considerable uncertainty about either or both variables [Paul *et al.*, 1991]. Hence, another approach monitors the performance of the algorithm and estimates the stopping point at run time [Horvitz, 1990; Zilberstein and Russell, 1995; Hansen and Zilberstein, 2001]. Given that the stopping point is continually revised based on the performance of the algorithm, this approach is called *monitoring*. Monitoring has proven to be a better approach to meta-level control than fixed allocation because it more effectively handles variance in the performance of the algorithm.

Existing meta-level control techniques that use either fixed allocation or monitoring have traditionally relied on significant offline work. In particular, before the start of an anytime algorithm, a model that describes its performance must be compiled for the given problem [Zilberstein, 1996]. Relying on such a model, called a *performance profile*, has several drawbacks. First, one must solve often more than a thousand instances of the problem to compile the model [Hansen and Zilberstein, 2001]. This costly overhead could take days or weeks for complex problems. Second, because each instance must be solved until completion, it can be infeasible to compile a model for intractable problems. Third, if one modifies the algorithm, the problem, or the system, the model must be recompiled. Fourth, since the model is an expectation over many instances, it may not accurately represent the performance of the algorithm on each specific instance. Finally, each instance must be drawn from a predicted distribution rather than the true distribution to compile the model. In short, existing techniques rely on substantial preprocessing that can decrease their practical utility and accuracy.

Addressing the drawbacks of existing meta-level control techniques, our primary contributions are: (1) an online performance prediction framework, (2) a meta-level control technique that uses online performance prediction, and (3) a myopic and nonmyopic projected stopping condition. Most importantly, we show that our approach outperforms existing techniques that require substantial offline work on several benchmark domains and a mobile robot domain.

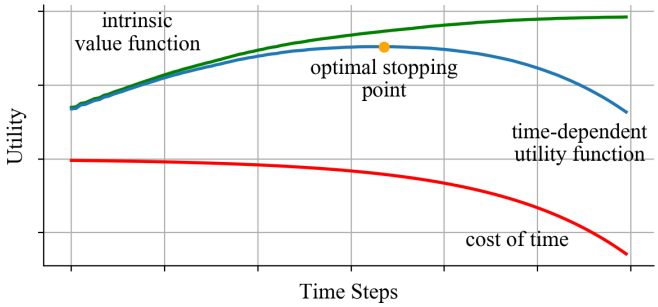

Figure 1: An idealized example of the meta-level control problem.

2 Meta-Level Control Problem

We begin by reviewing the meta-level control problem for anytime algorithms. This problem requires a model that represents the utility of a solution computed by an algorithm. Intuitively, in real-time decision-making tasks, a solution of a particular quality computed in a second has higher utility than a solution of the same quality computed in an hour. This suggests that the utility of a solution is a function of both quality and computation time [Horvitz and Rutledge, 1991; Boddy and Dean, 1994]. Accordingly, we define the utility of a solution as follows.

Definition 1. *A **time-dependent utility function**, $U(q,t)$, represents the utility of a solution of quality q at time step t.*

It is often possible to simplify a time-dependent utility function by expressing it as the difference between two functions called *object-level utility* and *inference-level utility* [Horvitz, 1988]. First, object-level utility represents the utility of a solution if we consider only the quality of that solution, ignoring the cost of computation time. Second, inference-level utility represents the utility of a solution if we take into account only the time needed to compute that solution, disregarding the value of solution quality. Adopting standard terminology [Russell and Wefald, 1991], we define this property below [Hansen and Zilberstein, 2001].

Definition 2. *A time-dependent utility function, $U(q,t)$, is **time-separable** if the utility of a solution of quality q at time step t can be expressed as the difference between two functions, $U(q,t) = U_I(q) - U_C(t)$, where $U_I(q)$ is the **intrinsic value function** and $U_C(t)$ is the **cost of time**.*

Given a time-dependent utility function, the meta-level control problem is the problem of deciding when to interrupt an anytime algorithm and act on the current solution. Figure 1 illustrates a typical instance of the meta-level control problem [Zilberstein, 1996]. In this example, the algorithm should be interrupted at the optimal stopping point. This is the point at which the time-dependent utility function is the highest. However, in practice, the optimal stopping point often cannot be determined due to considerable uncertainty about the performance of the algorithm or the urgency for the solution. The optimal stopping point must therefore be estimated using a model of either or both variables. Similar to earlier work [Hansen and Zilberstein, 2001], we assume that there is only uncertainty about the performance of the algorithm.

3 Online Performance Prediction

We now introduce an online performance prediction framework for anytime algorithms. Existing meta-level control techniques use performance profiles to predict the expected quality of the next solution as a function of several properties, including the computation time and the quality of the current solution [Dean and Boddy, 1988; Horvitz, 1990; Hansen and Zilberstein, 2001]. As discussed earlier, relying on a performance profile has several drawbacks in that it requires significant preprocessing that can decrease the practical utility and accuracy of meta-level control.

In place of a performance profile, we define a pair of vectors that jointly represent the performance of an anytime algorithm. The first vector describes the past performance of the algorithm as it solves a specific problem instance. Past performance can be expressed as a vector of solution qualities observed from the initial solution to the current solution. We formalize past performance below.

Definition 3. *A **performance history**, $\vec{h}$, represents the past performance of an anytime algorithm as a vector of solution qualities, $\vec{h} = [q_0\ q_1\ \ldots\ q_t]$, observed from the start time step to the current time step t at fixed intervals of Δt.*

The second vector represents the future performance of an anytime algorithm as it solves a specific problem instance. Future performance can be specified as a vector of solution qualities projected over the remaining time of the algorithm after the current solution to the final solution. We describe future performance as follows.

Definition 4. *A **performance projection**, $\vec{p}$, represents the future performance of an anytime algorithm as a vector of solution qualities, $\vec{p} = [q_{t+1}\ q_{t+2}\ \ldots\ q_T]$, projected from the time step $t+1$ to the final time step T at fixed intervals of Δt.*

Note that the final time step T is an upper bound on the desired time allocated to the algorithm. Unlike a performance profile, a performance projection is entirely based on the performance of the algorithm on the problem instance at hand.

To predict the future performance of an anytime algorithm, we use its past performance on the single problem instance being solved. This can be viewed as a function that computes a performance projection from a performance history. Note that this function can be implemented in many ways. In most cases, a simple method, such as linear or nonlinear regression, can compute a sensible performance projection from a performance history. It may also be possible to use richer models, such as neural networks or regression trees, that include features of the algorithm [Bartz-Beielstein and Markon, 2004; Huang *et al.*, 2010; Xu *et al.*, 2008]. However, these models must be adapted to an online context to avoid the drawbacks of a performance profile. Without committing to a specific implementation, we define this function broadly below.

Definition 5. *A **performance predictor**, $\Phi(\vec{h}) = \vec{p}$, maps a performance history $\vec{h}$ to a performance projection $\vec{p}$.*

We discuss a performance predictor that uses nonlinear regression later. Note that a performance predictor can compute performance projections from a weighted performance history with a bias toward recent solution qualities as well.

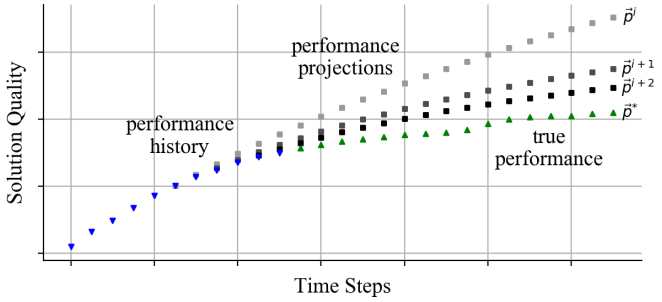

Figure 2: An illustration of online performance prediction.

Figure 2 offers an intuitive depiction of a performance predictor. Ideally, it computes performance projections that approach the true performance of an anytime algorithm as the size of the performance history increases. For instance, at the ith time step, the performance projection $\vec{p}^{\,i}$ does not closely approximate the true performance $\vec{p}^{\,*}$: in fact, it appears to be overly optimistic. However, at the $(i+1)$th time step, the next performance projection $\vec{p}^{\,i+1}$ draws closer to the true performance $\vec{p}^{\,*}$. In practice, as the performance predictor exploits more solution qualities in the performance history, the performance projections approach the true performance of an algorithm. Note that the performance history is a sequence of solution qualities observed from the start time step to the current time step of the algorithm. Similarly, each performance projection is a sequence of solution qualities projected after the current time step to the final time step of the algorithm.

4 Meta-Level Control Technique

In this section, we present a meta-level control technique for anytime algorithms that uses our online performance prediction framework. Similar to earlier work, our technique monitors the performance of the algorithm and estimates the stopping point at run time [Horvitz, 1990; Breese and Horvitz, 1991; Zilberstein and Russell, 1995; Hansen and Zilberstein, 2001]. However, unlike existing techniques that rely on a performance profile that must be compiled offline before the start of the algorithm, our technique uses online performance prediction: at every monitoring step, it computes a performance projection from a growing performance history using a performance predictor. Our technique therefore avoids relying on significant preprocessing that can decrease the practical utility and accuracy of meta-level control.

Algorithm 1 describes our meta-level control technique. First, the current time step and the performance history are initialized and the anytime algorithm is started. Note that the performance history is initially empty. Next, the performance of the algorithm is monitored at fixed intervals. At each monitoring step, the quality of the current solution is appended to the performance history and a performance projection is computed from that performance history using the performance predictor. If the performance projection has met the stopping condition, which we discuss in detail later, the algorithm is interrupted and the current solution is returned. Otherwise, the algorithm continues. The algorithm is monitored until interrupted or terminated naturally.

Algorithm 1: A meta-level control technique that uses online performance prediction.

Input: An anytime algorithm A, a performance predictor Φ, a projected stopping condition C, and a duration Δt
Output: A solution α

```
t ← 0
h⃗ ← [ ]
A.Start()
while A.Running() do
    α ← A.CurrentSolution()
    q ← α.Quality()
    h⃗ ← h⃗‖q
    p⃗ = Φ(h⃗)
    if C(p⃗) then
        A.Stop()
        return α
    t ← t + Δt
    Sleep(Δt)
return α
```

In general, a meta-level control technique uses a stopping condition to determine whether or not an anytime algorithm should be interrupted. If the stopping condition evaluates to true, the technique interrupts the algorithm. Otherwise, it lets the algorithm continue. An optimal stopping condition interrupts the algorithm when the *expected value of computation* (EVC) is no longer positive, where the EVC can be expressed as the expected improvement of the time-dependent utility of a solution [Horvitz, 1990]. To calculate the EVC, however, the technique must consider the entire sequence of remaining decisions about whether to continue or interrupt the algorithm. Thus, because such a calculation is often intractable, meta-level control uses an estimate of the EVC in practice.

Given this line of reasoning, our meta-level control technique uses a stopping condition that depends on the projected performance of an anytime algorithm. We call this a *projected stopping condition* and denote it as $C(\vec{p})$ in Algorithm 1. We define two projected stopping conditions below.

4.1 Myopic Projected Stopping Condition

The first stopping condition uses the projected one-step improvement of the utility of the current solution to determine whether or not the anytime algorithm should be interrupted. This can be expressed as the difference between two utilities: the utility of the projected *next* solution and the utility of the current solution. Note that these are both time-dependent utilities. We define the myopic improvement as follows.

Definition 6. *Suppose that an anytime algorithm computes a solution of quality q at time step t. The* ***myopic projected value of computation*** *(MPVC) is*

$$MPVC(q, t, \Delta t) = U(p_{t+\Delta t}, t + \Delta t) - U(q, t)$$

for an additional time step Δt given the current performance projection $\vec{p}$.

While the *MPVC* is positive, our technique lets the algorithm continue. Simply put, the algorithm continues so long

as the projected one-step improvement of the current solution is positive. We define our myopic stopping condition below.

Definition 7. *The meta-level control technique with the **myopic stopping condition** lets an anytime algorithm continue as long as MPVC*$(q, t, \Delta t) > 0$.

We refer to this version of our technique as the myopic meta-level control technique.

In practice, if the performance of the anytime algorithm is concave, our myopic technique will interrupt the algorithm at the optimal stopping point near the global maximum of the time-dependent utility. Intuitively, when the performance of the algorithm is concave, the benefit of continuing the algorithm diminishes over time. Thus, if our myopic technique interrupts the algorithm, that decision will remain optimal at any later time step. Often, however, the performance of the algorithm includes steps with little or no improvement. In this more likely case, because our myopic technique only considers the very next solution, it may interrupt the algorithm too early near a local maximum of the time-dependent utility. Hence, we define a more accurate yet more computationally demanding projected stopping condition that relaxes the assumption that the performance of the algorithm is concave.

4.2 Nonmyopic Projected Stopping Condition

The second stopping condition improves upon the first condition by considering the projected *best* solution instead of the projected *next* solution. Therefore, this can be expressed as the difference between two utilities: the utility of the projected *best* solution and the utility of the current solution. We define the nonmyopic improvement as follows.

Definition 8. *Suppose that an anytime algorithm computes a solution of quality q at time step t. The **nonmyopic projected value of computation** (NPVC) is*

$$NPVC(q, t) = max_{t'} U(p_{t'}, t') - U(q, t)$$

given the current performance projection $\vec{p}$.

While the *NPVC* is positive, our technique lets the algorithm continue. Intuitively, even if the projected next solution is worse, the algorithm continues so long as the projected future improvement of the current solution is positive. We define our nonmyopic stopping condition below.

Definition 9. *The meta-level control technique with the **nonmyopic stopping condition** lets an anytime algorithm continue as long as NPVC*$(q, t) > 0$.

Once again, we refer to this version of our technique as the nonmyopic meta-level control technique.

Our nonmyopic technique is not as shortsighted as our myopic technique. Simply put, because the nonmyopic stopping condition uses the projected best solution in place of the projected next solution, our nonmyopic technique is less likely to interrupt the anytime algorithm too early near a local maximum of the time-dependent utility. As a result, even when the performance of the algorithm includes steps with little or no improvement, our nonmyopic technique will still interrupt the algorithm closer to the optimal stopping point near the global maximum of the time-dependent utility. This results in more effective meta-level control.

5 Experiments

We compare our myopic and nonmyopic meta-level control technique to two state-of-the-art techniques developed by Hansen and Zilberstein [2001]:

1. a myopic monitor that interrupts an anytime algorithm once an estimate of the EVC is no longer positive, and
2. a nonmyopic monitor that interrupts an anytime algorithm once instructed to by a monitoring policy.

Note that these techniques rely on a performance profile that must be compiled offline prior to the activation of the anytime algorithm. The nonmyopic technique depends on a monitoring policy that must be computed offline as well.

All experiments represent a typical instance of the meta-level control problem where an intelligent system must decide when to interrupt an anytime algorithm and act on the current solution. To do this, we run two processes in parallel. The *object-level process* uses an anytime algorithm to solve an instance of a particular problem. At the same time, the *meta-level process* uses a meta-level control technique to monitor and control the anytime algorithm at fixed intervals. The experiment concludes when the anytime algorithm is either interrupted or terminated naturally. Note that all techniques monitor approximately every tenth of a second.

As discussed earlier, meta-level control requires a time-dependent utility function. Similar to earlier work, we define the time-dependent utility of a solution of quality q at time step t as the function, $U(q, t) = \alpha q - e^{\beta t}$, where the rates α and β are selected in practice based on the value of a solution and the urgency for a solution [Hansen and Zilberstein, 2001]. In all experiments, we deliberately select rates to avoid trivializing the problem (e.g., by making the urgency for a solution so high that the algorithm is interrupted immediately or so low that it runs to completion). Note that the first and second terms of the time-dependent utility function are the intrinsic value function and the cost of time respectively.

In contrast to existing techniques, our approach only requires a simple performance predictor. We use a performance predictor based on nonlinear regression (i.e., nonlinear least squares) with the model, $f(x; \vec{\theta}) = \theta_1 g(x + \theta_2) + \theta_3$, where the vector $\vec{\theta}$ contains the parameters of the model and the function g represents a nonlinear function. Because anytime algorithms generally exhibit the *diminishing returns* property [Zilberstein, 1996], many logarithmic and sigmoidal functions work effectively. Unlike a performance profile that must be built using a lengthy program, we emphasize that a performance predictor can be implemented in a few lines of code using the open-source Python library *SciPy*.

We apply all techniques to several benchmark domains and a mobile robot domain. Unless otherwise noted, we evaluate the performance of each technique using the *average time-dependent utility* across 50 instances of the problem. Thus, *higher* utilities indicate *better* performance. As an exception, when we have access to an efficient optimal solver, we use the *average time-dependent utility loss* instead. Hence, *lower* losses signify *better* performance. Note that the results of the myopic and nonmyopic techniques have been separated to guarantee a fair comparison.

Type	Prediction	50-TSP (%)	60-TSP (%)	70-TSP (%)	80-TSP (%)	90-TSP (%)
Nonmyopic	**Online**	**1.14 ± 0.16**	**1.64 ± 0.26**	**1.25 ± 0.20**	**1.41 ± 0.19**	**1.39 ± 0.26**
	Offline	1.81 ± 0.25	3.63 ± 0.47	2.53 ± 0.53	2.04 ± 0.26	3.65 ± 0.63
Myopic	**Online**	**14.34 ± 2.14**	**19.87 ± 4.18**	**29.22 ± 2.38**	**35.49 ± 1.80**	**37.43 ± 2.25**
	Offline	33.83 ± 2.64	46.50 ± 1.09	54.40 ± 1.07	57.47 ± 1.04	55.58 ± 1.10

Table 1: The average time-dependent *utility loss* for the best tour computed by the Lin-Kernighan heuristic on a range of TSPs.

Type	Prediction	20-JSP	40-JSP
Nonmyopic	**Online**	**119.19 ± 1.21**	**102.62 ± 0.82**
	Offline	115.64 ± 0.74	95.57 ± 0.25
Myopic	**Online**	**114.08 ± 1.92**	**97.49 ± 2.92**
	Offline	101.20 ± 0.81	91.28 ± 0.50

Table 2: The average time-dependent *utility* for the best schedule computed by the genetic algorithm on two JSPs.

Type	Prediction	100-QAP	200-QAP
Nonmyopic	**Online**	**165.55 ± 0.06**	**164.78 ± 0.04**
	Offline	162.78 ± 0.01	162.84 ± 0.04
Myopic	**Online**	**162.20 ± 0.74**	**160.77 ± 0.37**
	Offline	159.55 ± 0.08	159.32 ± 0.05

Table 3: The average time-dependent *utility* for the best assignment computed by simulated annealing on two QAPs.

5.1 Benchmark Domains

We begin by evaluating our meta-level control technique on several benchmark domains. Ideally, the quality of a solution can be defined as the approximation ratio, $q = c^*/c$, where c^* is the cost of the optimal solution and c is the cost of the current solution. However, because the cost of the optimal solution cannot quickly be computed for the benchmark problems, we estimate the quality of a solution as the approximation ratio, $q = \ell/c$, where ℓ is a problem-dependent lower bound on the optimal solution. For the performance predictor, we use the sigmoidal function $g(x) = \arctan(x)$ since the benchmark anytime algorithms exhibit diminishing returns.

Lin-Kernighan Heuristic

First, we consider a common benchmark in meta-level control of anytime algorithms [Hansen and Zilberstein, 2001] in Table 1. The Lin-Kernighan heuristic is a tour improvement algorithm that solves the traveling salesman problem (TSP) approximately [Lin and Kernighan, 1973]. The algorithm starts with an initial tour and gradually improves that tour by swapping specific subtours until convergence. We estimate solution (tour) quality using the lower bound, ℓ_{tsp}, defined as the length of the minimum spanning tree of the TSP.

Genetic Algorithms

Next, we examine a genetic algorithm that solves the job shop problem (JSP) approximately in Table 2. A JSP has a set of jobs composed of a sequence of tasks that must be scheduled on a set of machines. Since genetic algorithms have often been used to solve JSPs [Nakano and Yamada, 1991; Della Croce *et al.*, 1995], we use a standard open-source Python implementation, *jsp-ga*, for a genetic algorithm that includes swap mutation and generalized order crossover [Bierwirth, 1995; Puigcerver, 2013]. We estimate solution (schedule) quality using the lower bound, ℓ_{jsp}, defined as the time required to complete the longest job. Note that 30-JSP, 50-JSP, and 60-JSP show similar results.

Simulated Annealing

Finally, we explore a simulated annealing algorithm that solves the quadratic assignment problem (QAP) approximately in Table 3. A QAP has a set of facilities that must be assigned to a set of locations where a distance is given for each pair of locations and a flow is given for each pair of facilities. Since simulated annealing has often been used to solve QAPs [Misevičius, 2003], we use a standard open-source Fortran library, *QAPLIB*, for a simulated annealing algorithm [Burkard *et al.*, 1997]. We estimate solution (assignment) quality using the Gilmore-Lawler lower bound, ℓ_{qap}, which is the optimal cost of a linearization of a QAP [Gilmore, 1962]. Note that 50-QAP, 150-QAP, and 250-QAP show similar results.

5.2 Mobile Robot Domain

We now evaluate our meta-level control technique on a mobile robot domain. In simulation and on a mobile robot, we employ a path planning algorithm that generates paths that gradually minimize the probability of hitting obstacles. On an iClebo Kobuki, we use a standard open-source robotics C++ framework, *epic*, for the path planning algorithm [Wray *et al.*, 2016]. We measure solution (path) quality as just the probability of hitting obstacles. The mobile robot must therefore trade computation time with path safety. For the performance predictor, we use the logarithmic function $g(x) = \log(x)$ to demonstrate that other types of functions work effectively; however, we observe similar results with the sigmoidal function $g(x) = \arctan(x)$ as well. Conducting experiments on an actual robot ensures that our approach produces meaningful behavior and is suitable for use on real systems.

Table 4 shows the results of the experiments in simulation. In this case, we run all techniques on three maps. The first map, OFFICE, is a domain in which the goal is impeded by many boxes. The other maps, MINE-S and MINE-L, are well-known domains of coal mines generated using a mapping procedure developed by Thrun et al. [2003]. The instances of each problem are associated with a random start position but the same goal position.

Figure 3 depicts the results of the experiments on a mobile robot. In this case, we run our nonmyopic technique on the OFFICE map. We only consider our nonmyopic technique given its dominant performance in simulation. The four scenarios are associated with an infinite, high, low, and nil cost of time but the same start and goal position.

Type	Prediction	OFFICE	MINE-S	MINE-L
Nonmyopic	**Online**	**88.43 ± 0.67**	**87.90 ± 0.71**	**86.74 ± 1.16**
	Offline	79.12 ± 0.32	56.02 ± 0.16	70.83 ± 0.69
Myopic	**Online**	**85.60 ± 0.97**	**86.72 ± 0.83**	**84.62 ± 1.23**
	Offline	44.12 ± 3.84	52.91 ± 3.61	65.76 ± 3.60

Table 4: The average time-dependent *utility* for the best path computed by the path planning algorithm on three maps.

Figure 3: The OFFICE map (left) and the environment (right). The riskiest path (red), a very risky path (yellow), a very safe path (blue), and the safest path (green) are also shown.

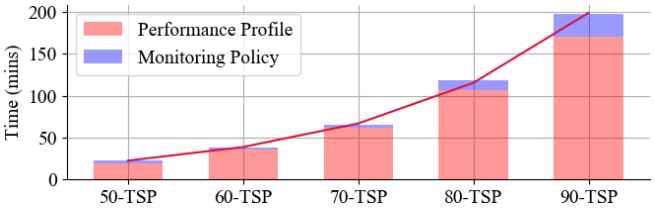

Figure 4: The preprocessing time of prevailing approaches.

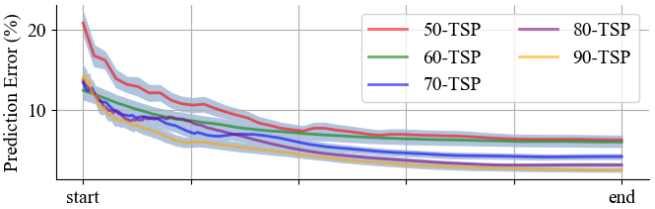

Figure 5: The change in the prediction error of our approach.

6 Discussion

On all benchmark domains, not only does our meta-level control technique avoid offline work, but it also outperforms the state-of-the-art. Given similar results across every domain, we focus our analysis on the Lin-Kernighan heuristic domain in Table 1. In the nonmyopic case, our technique incurs a loss under 2% on every problem. As the size of the problem increases, the loss of our nonmyopic technique remains steady while the existing technique varies. In the myopic case, the difference between our technique and the existing technique is even larger. As the size of the problem increases, the myopic technique degrades more slowly than the existing technique as well. While all techniques may be improved with a richer model using problem-specific instance features [Hutter *et al.*, 2014], it is encouraging that we obtain near optimal results given only computation time and solution quality.

Figure 4 illustrates the preprocessing time required to compile a performance profile and a monitoring policy for the state-of-the-art techniques on the Lin-Kernighan heuristic domain. This shows that preprocessing time grows rapidly with the size of the problem. In fact, even for modest problems, compiling a performance profile and a monitory policy can take hours of offline work. Furthermore, Figure 5 illustrates the improvement in the prediction error over time for our nonmyopic technique on the Lin-Kernighan heuristic domain. This shows that prediction error falls quickly with the size of the performance history. In particular, for all problems, the prediction error starts at less than 21% and ends at less than 7%. Note that the prediction error is expressed as the maximum difference between the current performance projection and the true performance of the anytime algorithm.

On the mobile robot domain, our technique substantially outperforms the state-of-the-art techniques. In simulation in Table 4, since the performance of the path planning algorithm is concave [Wray *et al.*, 2016], our myopic technique performs nearly as well as our nonmyopic technique. In fact, our myopic technique even outperforms the existing nonmyopic technique due to large variation across instances of the problem, which is not captured by a performance profile. Crucially, on a mobile robot in Figure 3, our nonmyopic technique effectively trades computation time with path safety. Given a high cost of time, the robot dangerously traverses through the boxes to the goal. However, given a low cost of time, the robot safely avoids the boxes entirely.

It may seem counterintuitive that our technique outperforms state-of-the-art techniques shown to be optimal [Hansen and Zilberstein, 2001]. The key idea is that existing techniques assume that a performance profile is an *exact* model of the behavior of an anytime algorithm. There are a number of reasons, however, why such a model may not be adequate. First, existing techniques assume that the model is accurate across different instances of a problem while our technique adapts to each instance by using online performance prediction. Next, when existing techniques are deployed, they do not perform as well because the model was compiled using some predicted distribution instead of the true but unknown distribution. Moreover, the model loses information about the performance of the algorithm as solution quality and computation time must be discretized into a small number of bins. Finally, since the model is compiled under specific CPU and memory conditions, existing techniques become less accurate given variance in these conditions. Even without any offline work, our technique avoids these pitfalls.

7 Conclusion

We offer a novel approach to meta-level control designed for practical and accurate use in intelligent systems. It offers many advantages over existing techniques: not only does it provide more effective meta-level control, but it also eliminates the need for any offline preprocessing. More importantly, to highlight that our approach produces effective meta-level control on the fly, we show that it outperforms state-of-the-art techniques on several benchmark domains and an actual mobile robot. Future work will explore more sophisticated performance predictors that use a richer, featured-based representation of the state of the anytime algorithm.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. We also thank Alan Labouseur and Sandhya Saisubramanian for their valuable feedback. This work was supported in part by the National Science Foundation grants IIS-1405550 and IIS-1724101.

References

[Bartz-Beielstein and Markon, 2004] Thomas Bartz-Beielstein and Sandor Markon. Tuning search algorithms for real-world applications: A regression tree based approach. In *IEEE Congress on Evolutionary Computation*, pages 1111–1118, 2004.

[Bierwirth, 1995] Christian Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations Research Spectrum*, 17(2-3):87–92, 1995.

[Boddy and Dean, 1994] Mark Boddy and Thomas L. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–285, 1994.

[Breese and Horvitz, 1991] John S. Breese and Eric J. Horvitz. Ideal reformulation of belief networks. In *6th Conf. on Uncertainty in Artificial Intelligence*, pages 129–143, 1991.

[Burkard *et al.*, 1997] Rainer E. Burkard, Stefan E. Karisch, and Franz Rendl. QAPLIB–A quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391–403, 1997.

[Dean and Boddy, 1988] Thomas L. Dean and Mark S. Boddy. An analysis of time-dependent planning. In *7th AAAI Nat'l Conf. on Artificial Intelligence*, pages 49–54, 1988.

[Della Croce *et al.*, 1995] Federico Della Croce, Roberto Tadei, and Giuseppe Volta. A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1):15–24, 1995.

[Gilmore, 1962] Paul C. Gilmore. Optimal and suboptimal algorithms for the quadratic assignment problem. *Journal of the Society for Industrial and Applied Mathematics*, 10(2):305–313, 1962.

[Hansen and Zilberstein, 2001] Eric A. Hansen and Shlomo Zilberstein. Monitoring and control of anytime algorithms: A dynamic programming approach. *Artificial Intelligence*, 126(1-2):139–157, 2001.

[Hansen *et al.*, 1997] Eric A. Hansen, Shlomo Zilberstein, and Victor A. Danilchenko. Anytime heuristic search: First results. Technical Report 97-50, Computer Science Department, University of Massachussetts Amherst, 1997.

[Horvitz and Rutledge, 1991] Eric Horvitz and Geoffrey Rutledge. Time-dependent utility and action under uncertainty. In *7th Conf. on Uncertainty in Artificial Intelligence*, pages 151–158, 1991.

[Horvitz, 1987] Eric J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *3rd Workshop on Uncertainty in Artificial Intelligence*, 1987.

[Horvitz, 1988] Eric Horvitz. Reasoning under varying and uncertain resource constraints. In *7th AAAI Nat'l Conf. on Artificial Intelligence*, pages 111–116, 1988.

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

[Huang *et al.*, 2010] Ling Huang, Jinzhu Jia, Bin Yu, Byung-Gon Chun, Petros Maniatis, and Mayur Naik. Predicting execution time of computer programs using sparse polynomial regression. In *24th Conf. on Neural Information Processing Systems*, pages 883–891, 2010.

[Hutter *et al.*, 2014] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[Lin and Kernighan, 1973] Shen Lin and Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[Luna *et al.*, 2013] R. Luna, I. A. Şucan, M. Moll, and L. E. Kavraki. Anytime solution optimization for sampling-based motion planning. In *IEEE Int'l Conf. on Robotics and Automation*, pages 5068–5074, 2013.

[Misevičius, 2003] Alfonsas Misevičius. A modified simulated annealing algorithm for the quadratic assignment problem. *Informatica*, 14(4):497–514, 2003.

[Nakano and Yamada, 1991] Ryohei Nakano and Takeshi Yamada. Conventional genetic algorithm for job shop problems. In *4th Int'l Conf. on Genetic Algorithms*, pages 474–479, 1991.

[Paul *et al.*, 1991] C. J. Paul, Anurag Acharya, Bryan Black, and Jay K. Strosnider. Reducing problem-solving variance to improve predictability. *Communications of the ACM*, 34(8):80–93, 1991.

[Pineau *et al.*, 2003] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *18th Int'l Joint Conf. on Artificial Intelligence*, pages 1025–1032, 2003.

[Puigcerver, 2013] Joan Puigcerver. jsp-ga. GitHub Repository, https://github.com/jpuigcerver/jsp-ga, 2013.

[Ramos and Cozman, 2005] "Fabio Tozeto Ramos and Fabio Gagliardi Cozman. Anytime anyspace probabilistic inference. *Int'l Journal of Approximate Reasoning*, 38(1):53 – 80, 2005.

[Richter *et al.*, 2010] Silvia Richter, Jordan Tyler Thayer, and Wheeler Ruml. The joy of forgetting: Faster anytime search via restarting. In *20th Int'l Conf. on Automated Planning and Scheduling*, pages 137–144, 2010.

[Richtsfeld *et al.*, 2013] Andreas Richtsfeld, Michael Zillich, and Markus Vincze. Anytime perceptual grouping of 2D features into 3D basic shapes. In *9th Int'l Conf. on Computer Vision Systems*, pages 73–82, 2013.

[Russell and Wefald, 1991] Stuart Russell and Eric Wefald. Principles of metareasoning. *Artificial Intelligence*, 49:361–395, 1991.

[Thrun *et al.*, 2003] Sebastian Thrun, Dirk Hahnel, David Ferguson, Michael Montemerlo, Rudolph Triebel, Wolfram Burgard, Christopher Baker, Zachary Omohundro, Scott Thayer, and William Whittaker. A system for volumetric robotic mapping of abandoned mines. In *IEEE Int'l Conf. on Robotics and Automation*, pages 4270–4275, 2003.

[Wray *et al.*, 2016] Kyle Hollins Wray, Dirk Ruiken, Roderic A. Grupen, and Shlomo Zilberstein. Log-space harmonic function path planning. In *IEEE Int'l Conf. on Intelligent Robots and Systems*, pages 1511–1516, 2016.

[Xu *et al.*, 2008] Lin Xu, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Satzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606, 2008.

[Zilberstein and Russell, 1995] Shlomo Zilberstein and Stuart J. Russell. Approximate reasoning using anytime algorithms. In S. Natarajan, editor, *Imprecise and Approximate Computation*, pages 43–62. Springer, 1995.

[Zilberstein, 1996] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73, 1996.