A Model-Free Approach to Meta-Level Control of Anytime Planning

Justin Svegliato[1], Prakhar Sharma[1], and Shlomo Zilberstein[1]

***Abstract*— Anytime algorithms offer a trade-off between solution quality and computation time that has proven to be useful in autonomous systems for a wide range of real-time planning problems. In order to optimize this trade-off, an autonomous system has to solve a challenging meta-level control problem: it must decide when to interrupt the anytime algorithm and act on the current solution. Current meta-level control techniques, however, make a number of unrealistic assumptions that reduce their effectiveness and usefulness in the real world. Eliminating these assumptions, we first introduce a model-free approach to meta-level control based on reinforcement learning and prove its optimality. We then offer a general meta-level control technique that can use different reinforcement learning methods. Finally, we show that our approach is effective across several common benchmark domains and a mobile robot domain.**

I. Introduction

Autonomous systems use anytime algorithms in many real-time planning problems, including motion planning [1], heuristic search [2], [3], object detection [4], belief space planning [5], [6], and probabilistic inference [7]. The central property of an anytime algorithm is that it can be interrupted at any time to provide a solution that is gradually improved upon during execution. This offers a trade-off between solution quality and computation time that has proven to be useful in autonomous systems that need to produce effective action in a timely manner. However, in order to optimize this trade-off, an autonomous system has to solve a challenging meta-level control problem: it must decide when to interrupt the anytime algorithm and act on the current solution.

There have been two main approaches to meta-level control for anytime algorithms in autonomous systems. The earliest approach that was proposed executes the anytime algorithm until its stopping point determined prior to runtime is reached [8], [9]. Because the stopping point is not adjusted once the anytime algorithm starts, this approach is called *fixed allocation*. While fixed allocation can be effective when there is little uncertainty in the performance of the anytime algorithm or the urgency for the solution, there is often substantial uncertainty in either or both variables in real-time planning problems [10]. In response, a more sophisticated approach that was proposed tracks the performance of the anytime algorithm and estimates its stopping point at runtime periodically [11], [12], [13], [14]. Since the stopping point is adjusted as the anytime algorithm runs, this approach is called *monitoring and control*. In general, monitoring and control has continually been shown to be a more effective approach to meta-level control than fixed allocation.

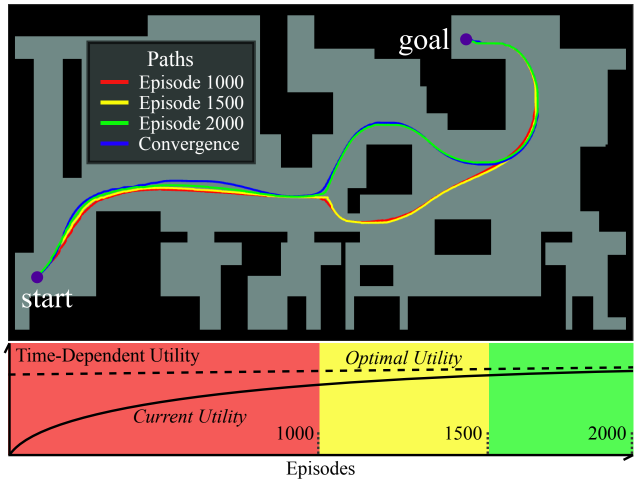

Fig. 1. An autonomous system can use a model-free approach to meta-level control based on reinforcement learning to learn when to interrupt a path planning algorithm and act on the current path plan. As the number of episodes increases, the utility of the current path plan approaches the utility of the final path plan that optimizes the trade-off between solution quality and computation time. Note that the path plan at convergence requires minutes while the final path plan only involves seconds of deliberation.

Current meta-level control techniques that monitor and control anytime algorithms have traditionally relied on planning with a model, called a *performance profile*, that describes the performance of the anytime algorithm solving a specific problem on a particular system. This model must be compiled offline before the activation of meta-level control by using the anytime algorithm to solve thousands of instances of the problem on the system. Planning with a model, however, imposes many assumptions that are often violated by autonomous systems that operate in real environments:

- *There must be enough time for offline compilation of the performance profile of the anytime algorithm.*
- *The settings of the anytime algorithm across every problem instance must be the same.*
- *The distribution of problem instances solved by the anytime algorithm must be known and fixed.*
- *The CPU and memory conditions of the system that executes the anytime algorithm must be static.*

Current techniques therefore make many assumptions that reduce their effectiveness and usefulness in the real world.

We propose a novel form of metareasoning illustrated in Fig. 1 that eliminates these unrealistic assumptions by: (1) introducing a model-free approach to meta-level control based on reinforcement learning and proving its optimality, (2) offering a general meta-level control technique that can use different reinforcement learning methods, and (3) showing that our approach is effective across several common benchmark domains and a mobile robot domain.

[1] College of Information and Computer Sciences, University of Massachusetts, Amherst, MA, USA. Emails: {jsvegliato, prakharsharm, shlomo}@cs.umass.edu

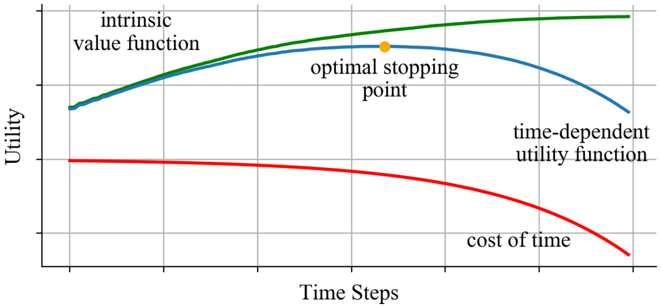

Fig. 2. An idealized example of the meta-level control problem

II. Meta-Level Control Problem

We begin by reviewing the meta-level control problem for anytime algorithms. This problem requires a model that describes the utility of a solution computed by an anytime algorithm. Naturally, in real-time planning problems, a solution of a specific quality calculated in a minute has lower utility than a solution of the same quality calculated in a second. At the minimum, this implies that the utility of a solution is likely a function of not only its quality but also its computation time [15], [9]. Following this line of intuition, we define the utility of a solution below.

Definition 1. *A **time-dependent utility function**,* $U : \Phi \times \Psi \to \mathbb{R}$*, represents the utility of a solution of quality* $q \in \Phi$ *at time step* $t \in \Psi$*.*

A time-dependent utility function can often be simplified by expressing it as the difference between two functions typically referred to as *object-level utility* and *inference-level utility* [16]. Object-level utility represents the utility of a solution by considering its quality but disregarding its computation time while inference-level utility represents the utility of a solution by taking into account its computation time but ignoring its quality. Adopting more recent terminology [17], we define such a property as follows [13].

Definition 2. *A time-dependent utility function,* $U : \Phi \times \Psi \to \mathbb{R}$*, is **time-separable** if the utility of a solution of quality* $q \in \Phi$ *at time step* $t \in \Psi$ *can be expressed as the difference between two functions,* $U(q, t) = U_I(q) - U_C(t)$*, where* $U_I : \Phi \to \mathbb{R}^+$ *is the **intrinsic value function** and* $U_C : \Psi \to \mathbb{R}^+$ *is the **cost of time**.*

Given a time-dependent utility function, the meta-level control problem is the problem in which an autonomous system must decide when to interrupt an anytime algorithm and act on the current solution. Fig. 2 provides an illustration of the meta-level control problem [18]. In this illustration, the algorithm should ideally be interrupted at the optimal stopping point because this is the maximum point of the time-dependent utility function. In practice, however, the optimal stopping point can rarely be determined as a result of considerable uncertainty about the performance of the algorithm or the urgency for the solution. The optimal stopping condition must therefore be approximated using an approach that models either or both variables. Note that we assume there is only uncertainty about the performance of the algorithm similar to earlier work [13].

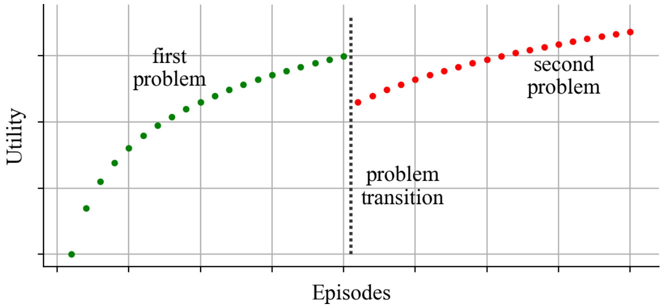

Fig. 3. An idealized illustration of model-free meta-level control

III. Model-Free Meta-Level Control

We now introduce a model-free approach to meta-level control based on reinforcement learning. Reinforcement learning has led to many methods [19] that have been effective across a wide range of applications from game playing [20] to helicopter control [21]. So as to maximize a notion of cumulative reward, a reinforcement learning agent learns how to operate in an environment *incrementally online* through reward signals in the form a policy. This is essential to meta-level control for two reasons. First, because there is often not enough time before the activation of meta-level control, the policy must be compiled online. Second, since the parameters of meta-level control often change over time, the policy must be updated incrementally. Reinforcement learning is therefore a natural approach to meta-level control.

Fig. 3 illustrates how a meta-level control technique that uses a reinforcement learning method learns its policy for a problem with parameters that change over time. Suppose the technique compiles its policy for the problem online (the *green* section of the *first* problem). When the parameters of the problem change (the *problem transition*), the policy of the technique may degrade in performance. In response, the technique updates its policy for the problem incrementally (the *red* section of the *second* problem). Generally, if there is not enough time before the activation of meta-level control or the parameters of meta-level control change over time, the technique can learn its policy on the fly from scratch.

Although we are not aware of any model-free approach to meta-level control based on reinforcement learning, such a framework is an especially good fit for several reasons. First, although the transition dynamics given the performance of an anytime algorithm may be unknown, reinforcement learning can learn an effective policy by balancing exploitation with exploration. Next, reinforcement learning can reduce the overhead of learning an effective policy by learning a *partial* policy that covers only reachable regions of the state space instead of a *universal* policy that covers the entire state space. This is because large regions of the state space are unlikely to be reached and can be ignored in practice. Finally, while the transition dynamics given the performance of an anytime algorithm may be nonstationary, reinforcement learning can maintain an effective policy by making adjustments with negligible overhead. In short, meta-level control shares a number of properties with problems for which reinforcement learning has been shown to be effective already.

Our model-free approach to meta-level control based on reinforcement learning expresses the meta-level control problem as a *Markov decision process* (MDP). An MDP is a formal decision-making model for reasoning in fully observable, stochastic environments that can be defined by a tuple $\langle S, A, T, R, s_0 \rangle$, where S is a finite set of states, A is a finite set of actions, $T : S \times A \times S \to [0, 1]$ represents the probability of reaching state $s' \in S$ after performing $a \in A$ in state $s \in S$, and $R : S \times A \times S \to \mathbb{R}$ represents the expected immediate reward of reaching state $s' \in S$ after performing action $a \in A$ in state $s \in S$, and $s_0 \in S$ is an optional start state. A solution to an MDP is a policy $\pi : S \to A$ that indicates that action $\pi(s) \in A$ should be performed in state $s \in S$. A policy π induces the value function $V^\pi : S \to \mathbb{R}$ that represents the expected cumulative reward $V^\pi(s)$ of each state $s \in S$. An optimal policy π^* maximizes the expected cumulative reward. There is no discount factor because the meta-level control problem has an indefinite horizon given that the anytime algorithm is always either interrupted or terminated naturally [22].

We now provide a formal description of the meta-level control problem by extending an MDP below.

Definition 3. *The **meta-level control problem** can be represented by a tuple $\langle \Phi, \Psi, S, A, T, R, s_0 \rangle$, where*

- *Φ is a set of qualities,*
- *Ψ is a set of time steps,*
- *$S = \Phi \times \Psi$ is a set of states of computation: each state $s \in S$ indicates that the anytime algorithm has a solution of quality $q \in \Phi$ at time step $t \in \Psi$,*
- *$A = \{\text{STOP}, \text{CONTINUE}\}$ is a set of actions such that* STOP *interrupts the anytime algorithm and* CONTINUE *executes the anytime algorithm for a time step,*
- *$T : S \times A \times S \to [0, 1]$ is an unknown transition function that describes the performance of the anytime algorithm solving an instance of a problem on a system,*
- *$R : S \times A \times S \to \mathbb{R}$ is a reward function called the **reward of anytime computation**, and*
- *$s_0 \in S$ is an optional start state that is typically $(0, 0)$.*

Note that any state $s \in S$ representing a solution of quality $q \in \Phi$ at time step $t \in \Psi$ can be denoted by a tuple (q, t).

Given that the transition function is unknown, the meta-level control problem has a reward function that describes the reward generated by using the anytime algorithm. This can be represented as a piecewise function of two components. If the action was to execute the anytime algorithm for a time step, the reward is the difference between the utility of the current solution and the utility of the previous solution. However, if the action was to interrupt the anytime algorithm, the reward is nil. We define the reward function as follows.

Definition 4. *Given a state $s = (q, t) \in S$, an action $a \in A$, and a successor state $s' = (q', t') \in S$, the **reward of anytime computation** can be represented by a function*

$$R(s, a, s) = \begin{cases} U(q', t') - U(q, t), & \text{if } a = \text{CONTINUE}, \\ 0, & \textit{otherwise}, \end{cases}$$

where $U : \Phi \times \Psi \to \mathbb{R}$ is a time-dependent utility function.

Algorithm 1: A general meta-level control technique that can use different reinforcement learning methods

```
Input: An anytime algorithm $\Lambda$, an action-value function $Q$,
       an update rule $\rho$, an exploration strategy $\xi$, and a
       duration $\Delta t$
Output: A solution $\sigma$
$s = (q, t) \leftarrow s_0$
$a \leftarrow \pi^Q_\xi(s)$
$\Lambda.\textit{Start}()$
$\textit{Sleep}(\Delta t)$
while $\Lambda.\textit{Running}()$ do
    $\sigma \leftarrow \Lambda.\textit{CurrentSolution}()$
    $s' = (q', t') \leftarrow (\sigma.\textit{Quality}(), t + \Delta t)$
    $r \leftarrow R(s, a, s')$
    $\rho(Q, r, \alpha)$
    $a \leftarrow \pi^Q_\xi(s')$
    if $a = \text{STOP}$ then
        $\Lambda.\textit{Stop}()$
        return $\sigma$
    $s \leftarrow s'$
    $\textit{Sleep}(\Delta t)$
return $\sigma$
```

It is easy to verify that the reward of anytime computation is consistent with the objective of optimizing time-dependent utility. Running the anytime algorithm until a solution of quality $q \in \Phi$ at time step $t \in \Psi$ results in a cumulative reward equal to the time-dependent utility $U(q, t)$.

An important goal of the meta-level control problem is to have an optimal policy that results in optimal meta-level control of the anytime algorithm given certain assumptions.

Theorem 1. *If the change in the quality of the solution over time is Markovian given its current quality $q \in \Phi$ and time step $t \in \Psi$, the optimal policy $\pi^* : S \to A$ of the meta-level control problem interrupts the anytime algorithm optimally.*

Proof (Sketch) 1. *This follows from the Markov assumption: the transition dynamics over the successor states of computation only depend on the current state of computation.*

Note that the optimal policy can be calculated or approximated using either planning or reinforcement learning.

Although many approaches to meta-level control have traditionally represented the state of computation as the quality and time step of a solution, such a representation may not be sufficient since the change in the quality of the solution given its current quality and time step may not be Markovian. This representation could therefore benefit from additional features that describe the state of computation. In particular, it could include algorithm-specific features, such as the size of the open list of Anytime A* [23], problem-specific features, such as the cluster distance of a TSP [24], or features specific to the underlying system. While our model-free approach has near optimal performance and fast convergence empirically, it can naturally augment the state of computation by using reinforcement learning.

IV. Meta-Level Control Technique

In this section, we offer a general meta-level control technique that can use different reinforcement learning methods. Similar to earlier work, our technique is a form of monitoring and control that tracks the performance of the anytime algorithm and estimates its stopping point at runtime periodically [11], [12], [13], [14]. However, while existing techniques rely on planning with a performance profile that must be compiled offline before the activation of meta-level control, our technique uses reinforcement learning to learn the policy incrementally online instead: it builds its policy gradually using the reward of anytime computation each time the anytime algorithm updates its solution to the instance of the problem. Therefore, by replacing offline compilation with online learning, our technique eliminates the unrealistic assumptions of existing techniques that reduce their effectiveness and usefulness in the real world.

Algorithm 1 outlines the general form of our technique. First, the state is initialized using the initial quality and time step, the action is initialized using the policy induced by the initial action-value function and the exploration strategy, and the anytime algorithm is started for a fixed duration. Next, the performance of the anytime algorithm is monitored at fixed intervals. During each monitoring step, the current solution is first retrieved from the anytime algorithm. The successor state is then built using the new quality and time step. The reward of anytime computation is subsequently calculated using the state, the action, and the successor state. The action-value function is in turn updated using the update rule based on the reward of anytime computation and the learning rate. An action is once again selected from the policy induced by the updated action-value function and the exploration strategy. Finally, if the action indicates to stop, the anytime algorithm is interrupted and the current solution is returned. Otherwise, the state is set to the successor state and the anytime algorithm continues to run for a fixed duration. The anytime algorithm is monitored at fixed intervals until it is interrupted or terminated naturally. Note that the action-value function easily can be represented by a table [19] or approximated by a linear [25] or nonlinear function [26].

Our technique has been generalized to support many reinforcement learning methods. In particular, it can use on-policy and off-policy temporal difference (TD) learning methods like TD(λ) and SARSA(λ) [27], [28] in addition to exploration strategies like ϵ-greedy and softmax action selection [19]. Although we do not commit to a specific reinforcement learning method, we describe our technique using *ϵ-greedy Q-learning* [29] as an example because it has been analyzed extensively and shown to be effective across many applications [30], [31], [32]. We discuss the update rules and the exploration strategies of our technique below.

A. Update Rules

Any reinforcement learning agent updates its action-value function by following an *update rule* that uses a reward signal emitted by the environment. In Algorithm 1, when a new solution is computed in each monitoring step, our technique updates the action-value function using the update rule based on the reward of anytime computation and the learning rate. When the anytime algorithm is interrupted, however, our technique does not update the action-value function because there is no change in the solution.

ϵ-greedy Q-learning Example: Given an action-value function Q, a reward of anytime computation $r = R(s, a, s')$, and a learning rate α, the update rule $\rho(Q, r, \alpha)$ is below:

$$Q(s,a) \overset{+}{\leftarrow} \alpha[r + \max_{a' \in A} Q(s', a') - Q(s, a)],$$

where the current state is $s = (q, t) \in S$, the current action is $a \in A$, and the successor state is $s' = (q', t') \in S$.

B. Exploration Strategies

Any reinforcement learning agent balances exploitation with exploration by following an *exploration strategy*. In Algorithm 1, when the action-value function is updated in each monitoring step, our technique updates the policy using the action-value function and the exploration strategy.

ϵ-greedy Q-learning Example: The greedy policy must first be calculated. This policy can be built by performing a one-step lookahead over every action available at the current state. Given an action-value function Q, the greedy policy $\pi^Q(s)$ is calculated below:

$$\pi^Q(s) \leftarrow \arg\max_{a \in A} Q(s, a),$$

where the current state is $s = (q, t) \in S$. Finally, once the greedy policy has been calculated, it can be modified to follow ϵ-greedy exploration by introducing randomness. Given an exploration probability ϵ and a greedy policy π^Q, the ϵ-greedy policy $\pi^Q_\xi(s)$ is calculated below:

$$\pi^Q_\xi(s) = \begin{cases} \pi^Q(s), & \textit{with probability 1 - } \epsilon, \\ \text{random}(A), & \textit{otherwise}, \end{cases}$$

where the current state is $s = (q, t) \in S$.

V. Experiments

We compare our model-free approach to meta-level control based on reinforcement learning to the prevailing general-purpose planning technique that can be used with any anytime algorithm [13]. Each version of our technique uses a different reinforcement learning method with some function representation following ϵ-greedy action selection. In particular, we evaluate the following versions of our technique: tabular SARSA, tabular Q-learning, Fourier basis SARSA, and Fourier basis Q-learning. Note that a tabular function and a linear approximation with Fourier basis are often the first to be tried by reinforcement learning methods [25].

All meta-level control techniques have been evaluated on several common benchmark domains and a mobile robot domain. In each domain, an autonomous system solves the meta-level control problem for a given anytime algorithm on a specific problem. To do this, each trial runs two processes in parallel. The *object-level process* solves an instance of the problem with the anytime algorithm while the *meta-level*

TABLE I

THE LIN-KERNIGHAN HEURISTIC DOMAIN RESULTS

Method	40-TSP (%)	50-TSP (%)	60-TSP (%)	70-TSP (%)	80-TSP (%)	90-TSP (%)
Planning	9.67 ± 0.63	11.40 ± 0.79	15.27 ± 1.90	8.95 ± 0.64	10.68 ± 0.80	11.96 ± 0.85
SARSA(Table)	11.11 ± 2.76	13.94 ± 2.31	16.73 ± 0.91	13.61 ± 1.74	24.49 ± 1.36	21.46 ± 0.91
Q-learning(Table)	8.62 ± 2.42	15.66 ± 1.88	18.61 ± 1.13	20.93 ± 1.99	23.34 ± 1.62	26.44 ± 0.89
SARSA(Fourier)	3.84 ± 1.22	**2.27 ± 0.34**	6.77 ± 1.14	3.75 ± 0.94	**3.81 ± 0.44**	5.68 ± 0.53
Q-learning(Fourier)	**2.93 ± 0.91**	2.69 ± 0.37	**6.33 ± 1.36**	**2.51 ± 0.69**	5.64 ± 0.69	**5.25 ± 0.41**

TABLE II

THE GENETIC ALGORITHMS DOMAIN RESULTS

Method	20-JSP (%)	40-JSP (%)	60-JSP (%)
Planning	2.85 ± 0.47	5.54 ± 0.56	2.52 ± 0.52
SARSA(Table)	18.26 ± 0.42	17.23 ± 0.37	15.33 ± 0.31
Q-learning(Table)	18.17 ± 0.33	16.96 ± 0.37	14.43 ± 0.28
SARSA(Fourier)	**2.11 ± 0.32**	2.37 ± 0.55	**1.38 ± 0.34**
Q-learning(Fourier)	2.77 ± 0.36	**1.88 ± 0.27**	2.22 ± 0.55

TABLE III

THE SIMULATED ANNEALING DOMAIN RESULTS

Method	100-QAP (%)	150-QAP (%)	200-QAP (%)
Planning	4.33 ± 0.27	6.52 ± 0.24	7.13 ± 0.19
SARSA(Table)	4.13 ± 0.91	3.97 ± 0.13	4.39 ± 0.17
Q-learning(Table)	3.36 ± 0.94	3.95 ± 0.27	3.52 ± 0.53
SARSA(Fourier)	0.69 ± 0.15	**0.51 ± 0.11**	**1.12 ± 0.11**
Q-learning(Fourier)	**0.36 ± 0.21**	0.53 ± 0.21	1.17 ± 0.43

process monitors and controls the anytime algorithm with the meta-level control technique. The trial is over once the anytime algorithm is interrupted or terminated naturally.

Any meta-level control problem requires a time-dependent utility function. Following earlier work, given a solution of quality $q \in \Phi$ at time step $t \in \Psi$, the time-dependent utility can be defined as the function $U(q, t) = \alpha q - e^{\beta t}$, where the rates α and β are based on the value of a solution and the urgency for a solution [13]. These rates have been selected deliberately to avoid trivializing the problem by making the urgency for a solution so low that the anytime algorithm runs to completion or so high that it is interrupted immediately.

All of our meta-level control techniques begin with a randomized initial policy that is equally likely to stop or continue the anytime algorithm. This policy is updated as the technique learns from 5000 random problem instances. The exploration probability ϵ is set to 0.1 with a decay of 0.999 while the learning rate α is set to 0.1 for our tabular techniques and 0.00001 for our Fourier basis techniques. It is also possible to design the initial policy intelligently depending on its function representation by exploiting the form of the time-dependent utility function. The planning technique, however, uses a static policy that takes substantially longer to calculate since it applies dynamic programming to a performance profile compiled from 2000 random problem instances that are solved to *completion*. It therefore would have been infeasible to use 5000 random problem instances.

A. Common Benchmark Domains

We begin by evaluating our meta-level control approach on several common 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 any benchmark problem, 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.

All of our meta-level control technique are evaluated along three dimensions: the degree of optimality, the rate of convergence, and the rate of adaptation. First, for optimality, Tables I, II, and III show the average time-dependent utility loss of the final solution for each of our techniques across 100 instances of all benchmark problems. Next, for convergence, Fig. 4 shows the change in the time-dependent utility of the policy for each of our techniques on select benchmark problems. Finally, for adaptation, Fig. 5 shows the number of episodes required by each of our techniques to adapt to a change in the parameters of select benchmark problems.

1) Lin-Kernighan Heuristic Domain: The first domain uses the Lin-Kernighan heuristic to solve travelling salesman problems (TSP). A TSP has a set of cities that must be visited using the shortest possible route where a distance is given for each pair of cities. The Lin-Kernighan heuristic is a tour improvement algorithm that starts with an initial tour and gradually improves that tour by swapping specific subtours until convergence [33]. Solution (tour) quality is approximated using the length of the minimum spanning tree of the TSP as the lower bound ℓ_{tsp}.

2) Genetic Algorithms Domain: The next domain uses a genetic algorithm to solve job-shop problems (JSP). A JSP has a set of jobs composed of a sequence of tasks that must be scheduled on a set of machines. The genetic algorithm is a standard open-source Python implementation based on swap mutation and generalized order crossover used to solve JSPs approximately [34]. Solution (schedule) quality is approximated using the time required to complete the longest job as the lower bound ℓ_{jsp}.

3) Simulated Annealing Domain: The final domain uses simulated annealing to solve quadratic assignment problems (QAP). 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. The simulating annealing algorithm is a standard open-source Fortran implementation used to solve QAPs approximately [35]. Solution (assignment) quality is approximated using the Gilmore-Lawler bound, the optimal cost of a linearized QAP [36], as the lower bound ℓ_{qap}.

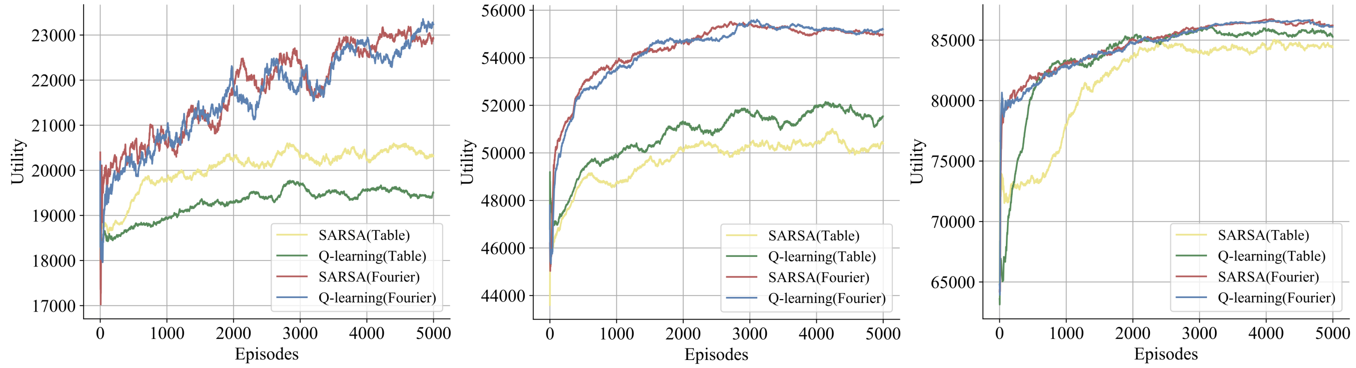

Fig. 4. The learning curves for each of our meta-level control techniques on the 60-Tsp, 40-Jsp, and 150-Qap benchmark problems

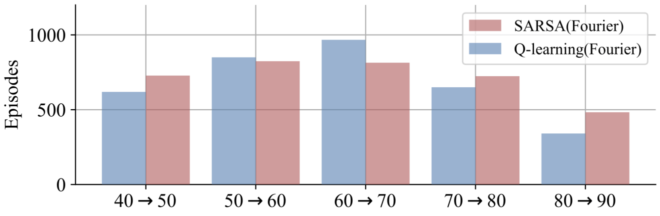

Fig. 5. The adaptation period for our Fourier basis meta-level control techniques on every TSP benchmark problem

B. Mobile Robot Domain

We now evaluate our meta-level control approach on a mobile robot domain. On an iClebo Kobuki in simulation, we use a path planning algorithm that computes path plans that minimize the probability of collision gradually from an open-source robotics C++ framework called *epic* [37]. Solution (path plan) quality is defined as safety in terms of the probability of collision. The mobile robot must therefore trade computation time with safety. Note that Fig. 1 is an illustration of the mobile robot domain in simulation.

All of our meta-level control techniques are evaluated on their degree of optimality. Table IV shows the average time-dependent utility loss of the final solution for each of our techniques across 100 instances of three path planning problems. Each problem uses a different map. The Office map is a room in which the goal is impeded by furniture. The Mine-S and Mine-L maps are coal mines generated by a standard mapping procedure [38]. All instances of each problem have a random start but the same goal position.

TABLE IV
The mobile robot domain results

Method	Office (%)	Mine-S (%)	Mine-L (%)
Planning	12.02 ± 0.22	10.64 ± 0.22	11.02 ± 0.18
SARSA(Table)	5.52 ± 0.36	6.72 ± 0.41	5.66 ± 0.26
Q-learning(Table)	3.59 ± 0.18	6.01 ± 0.23	4.08 ± 0.15
SARSA(Fourier)	2.95 ± 0.19	3.37 ± 0.14	**2.34 ± 0.75**
Q-learning(Fourier)	**2.75 ± 0.17**	**3.15 ± 0.15**	3.13 ± 0.25

VI. Discussion

Our model-free approach to meta-level control based on reinforcement learning outperforms the planning technique on every domain. Given near optimal performance in Tables I, II, III, and IV and fast convergence in Fig. 4, we focus on our techniques that use a linear approximation with Fourier basis. Our techniques incur a loss lower than 3% on most problems with an upper limit of 7% while the planning technique incurs a loss higher than 10% on most problems with an upper limit of 16%. Our techniques also have less variance compared to the planning technique. Overall, while our approach can be improved in many ways, it is encouraging that it has near optimal performance and fast convergence using standard reinforcement learning methods, simple function representations, and naive action selection.

Our approach also adapts to meta-level control problems with parameters that change over time. In Fig. 5, our techniques update their policies in under 1000 random instances to adapt to a change in the size of each TSP benchmark problem. The planning technique, however, requires substantial offline work because it has to compile a completely new policy by applying dynamic programming to a performance profile before the activation of meta-level control.

Using reinforcement learning for model-free meta-level control offers a number of advantages over the traditional planning paradigm. First, when the parameters of meta-level control change over time, our approach can update its policy incrementally on the fly. This is critical since the settings of the anytime algorithm, the distribution of problem instances, and the CPU and memory conditions of the system often shift in practice. Moreover, when there is not enough time before the activation of meta-level control, our approach can compile its policy online from scratch. Most importantly, even if the parameters of meta-level control do *not* change over time and there *is* enough time before the activation of meta-level control, our approach still outperforms the planning paradigm by learning a significantly *more effective* policy in substantially *less time*. This is because reinforcement learning focuses on meaningful regions of the state space of the meta-level control problem in contrast to planning.

VII. Conclusion

We propose a model-free approach to meta-level control based on reinforcement learning with a number of advantages over the traditional planning paradigm. It not only outperforms existing techniques but also relaxes the assumptions that reduce their effectiveness and usefulness in the real world. Future work will explore sophisticated reinforcement learning methods that use a neural network function approximation with a feature-based state of computation.

References

[1] S. Karayev, M. Fritz, and T. Darrell, “Anytime recognition of objects and scenes,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 572–579.

[2] E. Burns, W. Ruml, and M. B. Do, “Heuristic search when time matters,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 697–740, 2013.

[3] B. Cserna, W. Ruml, and J. Frank, “Planning time to think: Metareasoning for on-line planning with durative actions,” in *Proceedings of the 27th International Conference on Automated Planning and Scheduling*, 2017.

[4] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, “Anytime motion planning using the RRT*,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2011, pp. 1478–1483.

[5] J. Pineau, G. Gordon, and S. Thrun, “Point-based value iteration: An anytime algorithm for POMDPs,” in *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, 2003, pp. 1025–1032.

[6] M. T. Spaan and N. Vlassis, “Perseus: Randomized point-based value iteration for POMDPs,” *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, 2005.

[7] F. T. Ramos and F. G. Cozman, “Anytime anyspace probabilistic inference,” *Journal of Approximate Reasoning*, vol. 38, no. 1, 2005.

[8] E. J. Horvitz, “Reasoning about beliefs and actions under computational resource constraints,” in *Proceedings of the 3rd Workshop on Uncertainty in Artificial Intelligence*, 1987.

[9] M. Boddy and T. L. Dean, “Deliberation scheduling for problem solving in time-constrained environments,” *Artificial Intelligence*, vol. 67, no. 2, pp. 245–285, 1994.

[10] C. J. Paul, A. Acharya, B. Black, and J. K. Strosnider, “Reducing problem-solving variance to improve predictability,” *Communications of the ACM*, vol. 34, no. 8, pp. 80–93, 1991.

[11] E. J. Horvitz, “Computation and action under bounded resources,” Ph.D. dissertation, Stanford University, CA, 1990.

[12] S. Zilberstein and S. J. Russell, “Approximate reasoning using anytime algorithms,” in *Imprecise and Approximate Computation*, S. Natarajan, Ed. Springer, 1995, pp. 43–62.

[13] E. A. Hansen and S. Zilberstein, “Monitoring and control of anytime algorithms: A dynamic programming approach,” *Artificial Intelligence*, vol. 126, no. 1-2, pp. 139–157, 2001.

[14] C. H. Lin, A. Kolobov, E. Kamar, and E. Horvitz, “Metareasoning for planning under uncertainty,” in *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 2015.

[15] E. Horvitz and G. Rutledge, “Time-dependent utility and action under uncertainty,” in *Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence*, 1991, pp. 151–158.

[16] E. Horvitz, “Reasoning under varying and uncertain resource constraints.” in *Proceedings of the 7th AAAI Conference on Artificial Intelligence*, 1988, pp. 111–116.

[17] S. Russell and E. Wefald, “Principles of metareasoning,” *Artificial Intelligence*, vol. 49, pp. 361–395, 1991.

[18] S. Zilberstein, “Using anytime algorithms in intelligent systems,” *AI Magazine*, vol. 17, no. 3, p. 73, 1996.

[25] G. Konidaris, S. Osentoski, and P. S. Thomas, “Value function approximation in reinforcement learning using the Fourier basis.” in *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, vol. 6, 2011, p. 7.

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.

[20] G. Tesauro, “Temporal difference learning and TD-gammon,” *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[21] H. Kim, M. I. Jordan, S. Sastry, and A. Y. Ng, “Autonomous helicopter flight via reinforcement learning,” in *Proceedings of the Conference on Neural Information Processing Systems*, 2004, pp. 799–806.

[22] E. A. Hansen, “Indefinite-horizon POMDPs with action-based termination,” in *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, 2007, pp. 1237–1242.

[23] E. A. Hansen and R. Zhou, “Anytime heuristic search,” *Journal of Artificial Intelligence Research*, vol. 28, pp. 267–297, 2007.

[24] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, “Algorithm runtime prediction: Methods & evaluation,” *Artificial Intelligence*, vol. 206, pp. 79–111, 2014.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[27] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1995.

[28] S.-L. Chen and Y.-M. Wei, “Least-squares SARSA(Lambda) algorithms for reinforcement learning,” in *Proceedings of the 4th International Conference on Natural Computation*, 2008, pp. 632–636.

[29] C. J. C. H. Watkins and P. Dayan, “Q-learning,” *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[30] A. Srivihok and P. Sukonmanee, “E-commerce intelligent agent: Personalization travel support agent using Q-learning,” in *Proceedings of the 7th International Conference on Electronic Commerce*, 2005, pp. 287–292.

[31] Y. Tan, W. Liu, and Q. Qiu, “Adaptive power management using RL,” in *Proceedings of the International Conference on Computer-Aided Design*, 2009, pp. 461–467.

[32] H. R. Maei, C. Szepesvári, S. Bhatnagar, and R. S. Sutton, “Toward off-policy learning control with function approximation,” in *Proceedings of the 27th International Conference on Machine Learning*, 2010.

[33] S. Lin and B. W. Kernighan, “An effective heuristic algorithm for the traveling-salesman problem,” *Operations Research*, vol. 21, no. 2, pp. 498–516, 1973.

[34] C. Bierwirth, “A generalized permutation approach to job shop scheduling with genetic algorithms,” *Operations Research Spectrum*, vol. 17, no. 2-3, pp. 87–92, 1995.

[35] A. Misevičius, “A modified simulated annealing algorithm for the quadratic assignment problem,” *Informatica*, vol. 14, no. 4, pp. 497–514, 2003.

[36] P. C. Gilmore, “Optimal/suboptimal algorithms for the quadratic assignment problem,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, no. 2, pp. 305–313, 1962.

[37] K. H. Wray, D. Ruiken, R. A. Grupen, and S. Zilberstein, “Log-space harmonic function path planning,” in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, 2016, pp. 1511–1516.

[38] S. Thrun, D. Hahnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker, “A system for volumetric robotic mapping of abandoned mines,” in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2003, pp. 4270–4275.