Tuning the Hyperparameters of Anytime Planning: A Metareasoning Approach with Deep Reinforcement Learning

Abhinav Bhatia,[1] **Justin Svegliato,**[2] **Samer B. Nashed,**[1] **Shlomo Zilberstein**[1]

[1] College of Information and Computer Sciences, University of Massachusetts Amherst
[2] Department of Electrical Engineering and Computer Sciences, University of California Berkeley
abhinavbhati@cs.umass.edu, jsvegliato@berkeley.edu, snashed@cs.umass.edu, shlomo@cs.umass.edu

Abstract

Anytime planning algorithms often have hyperparameters that can be tuned at runtime to optimize their performance. While work on metareasoning has focused on when to interrupt an anytime planner and act on the current plan, the scope of metareasoning can be expanded to tuning the hyperparameters of the anytime planner at runtime. This paper introduces a general, decision-theoretic metareasoning approach that optimizes both the stopping point and hyperparameters of anytime planning. We begin by proposing a generalization of the standard meta-level control problem for anytime algorithms. We then offer a meta-level control technique that monitors and controls an anytime algorithm using deep reinforcement learning. Finally, we show that our approach boosts performance on a common benchmark domain that uses anytime weighted A* to solve a range of heuristic search problems and a mobile robot application that uses RRT* to solve motion planning problems.

Introduction

Anytime algorithms often have hyperparameters that can be tuned at runtime to optimize their performance in a given scenario—a specific problem instance and time constraint. Simply put, an anytime algorithm is an algorithm that gradually improves its current solution at runtime and can be interrupted at any time for that solution (Zilberstein 1996). This offers a trade-off between solution quality and computation time that has proven to be useful in real-time decision-making, such as motion planning (Karaman et al. 2011), heuristic search (Burns, Ruml, and Do 2013), object detection (Karayev, Fritz, and Darrell 2014), belief space planning (Spaan and Vlassis 2005), and probabilistic inference (Ramos and Cozman 2005). Naturally, to manage this trade-off, current work on metareasoning focuses on determining when to interrupt an anytime planner and act on the current plan. However, the scope of metareasoning can ideally be expanded to tuning the hyperparameters of an anytime planner at runtime to optimize its performance.

There has been substantial work on metareasoning that determines when to interrupt an anytime algorithm and act on the current solution. Generally, these methods monitor and control an anytime algorithm by tracking its performance and calculating its stopping point at runtime. For example, an early method estimates the optimal stopping point by solving a sequential decision problem with dynamic programming (Hansen and Zilberstein 2001) while a more recent method estimates the optimal stopping point by predicting its performance online (Svegliato, Wray, and Zilberstein 2018). These methods, however, cannot tune the hyperparameters of an anytime algorithm at runtime.

Methods for tuning the hyperparameters of an anytime algorithm at runtime have largely been designed for specific anytime algorithms and often lack formal analysis and generality while requiring expertise in the implementation.

In this paper, we introduce a general, decision-theoretic metareasoning approach for both optimal stopping and hyperparameter tuning of anytime planning. Our approach approximates the problem of monitoring and controlling an anytime algorithm as a Markov decision process (MDP). Its *states* represent the quality and computation time of the current solution and any other features needed to summarize the internal state of the algorithm, the instance of the problem, or the performance of the underlying system, while its *actions* represent either interrupting the algorithm or executing the algorithm for another time step while tuning its internal hyperparameters. Our approach solves this MDP using deep reinforcement learning (RL) to obtain a policy for both optimal stopping and hyperparameter tuning of the anytime algorithm: it performs a series of episodes, each of which applies the anytime algorithm to some generated instance of a specific problem. Our experiments on distinct problem paradigms—*anytime weighted A** for heuristic search and *RRT** for motion planning—show that deep RL is a natural approach to metareasoning for anytime algorithms since the performance of an anytime algorithm is generally hard to predict, but an abundance of simulations of the algorithm can be easily performed.

Our main contributions are: (1) a generalization of the standard meta-level control problem for anytime algorithms, (2) a meta-level control technique that monitors and controls an anytime algorithm based on deep RL, and (3) experiments showing that our approach boosts performance on a common benchmark domain that uses anytime weighted A* to solve a range of heuristic search problems and a mobile robot application that uses RRT* for motion planning.

Related Work

The literature on automated online hyperparameter tuning for general algorithms is vast. Commonly used methods include local search (Adenso-Diaz and Laguna 2006; Hutter, Hoos, and Stützle 2007; Hutter et al. 2009b), genetic algorithms (Ansótegui, Sellmann, and Tierney 2009), racing algorithms (Birattari et al. 2002, 2010), and sequential model-based optimization (Hutter et al. 2009a, 2010; Hutter, Hoos et al. 2011).

These methods have largely been designed for general algorithms and generally do not exploit anytime algorithms. By applying deep RL to anytime algorithms in particular, our approach avoids the drawbacks often imposed by current methods. First, unlike methods that only support numerical hyperparameters and deterministic algorithms, our approach supports both categorical hyperparameters and stochastic algorithms. Next, unlike methods that only optimize an algorithm's hyperparameters on a single known problem instance, our approach optimizes an algorithm's hyperparameters for each problem instance drawn from a *problem distribution*. In fact, our approach can tailor the algorithm's hyperparameters to a specific instance of a problem at runtime—making adjustments in a non-myopic, decision-theoretic manner. Finally, unlike methods that always execute an algorithm until completion, our approach learns to terminate an algorithm early when it is beneficial.

Only one other approach that we are aware of proposes using reinforcement learning for automated hyperparameter tuning (Biedenkapp et al. 2020). Similar to our approach, it represents the problem as an MDP to be solved using deep reinforcement learning. However, while it provides a template for general algorithms, our approach is tailored to anytime algorithms, optimizing both the stopping point and hyperparameters, with specific MDP characteristics and reward structure that maximizes the value of the final solution produced by the anytime algorithm prior to termination.

There has been a large body of work on optimal stopping for anytime algorithms. The earliest approach, namely *fixed allocation*, executes the algorithm until a stopping point determined prior to runtime (Horvitz 1987; Boddy and Dean 1994). While fixed allocation is effective given negligible uncertainty in the performance of the anytime algorithm, there is often substantial uncertainty in real-time planning (Paul et al. 1991). Hence, a more sophisticated approach, namely *monitoring and control*, tracks the performance of the algorithm and estimates a stopping point at runtime periodically (Horvitz 1990; Zilberstein and Russell 1995; Hansen and Zilberstein 2001; Lin et al. 2015; Svegliato, Wray, and Zilberstein 2018; Svegliato and Zilberstein 2018; Svegliato, Sharma, and Zilberstein 2020). Our approach not only determines the stopping point but also tunes the hyperparameters of an anytime algorithm at runtime.

Finally, there are specialized methods for heuristically tuning certain hyperparameters of anytime weighted A* (Hansen and Zhou 2007; Sun, Druzdzel, and Yuan 2007; Thayer and Ruml 2009; Bhatia, Svegliato, and Zilberstein 2021) and RRT* (Urmson and Simmons 2003; Akgun and Stilman 2011; Kiesel, Burns, and Ruml 2012), without optimizing the stopping point. In this work, we apply our approach to optimize both the stopping point and tune selected hyperparameters of anytime weighted A* and RRT* at runtime, experimenting with the vanilla versions of these algorithms for simplicity and generality as our approach focuses broadly on anytime algorithms instead of heuristic search or motion planning.

Standard Meta-Level Control Problem

We begin by reviewing the standard meta-level control problem for anytime algorithms. This requires a function that describes the utility of a solution computed by an anytime algorithm in terms of its quality and computation time (Horvitz and Rutledge 1991). We define this function below.

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

A time-dependent utility function can be expressed as the difference between an *intrinsic value function* that describes the utility of a solution given its quality but not its computation time and a *cost of time* that describes the utility of a solution given its computation time but not its quality (Horvitz 1988). We present this property as follows.

Definition 2. *A time-dependent utility function $U : \Phi \times \Psi \to \mathbb{R}$ is **time-separable** if the utility $U(q, t)$ 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**.*

The standard meta-level control problem for anytime algorithms is the problem of determining when to interrupt an anytime algorithm and act on the current solution (Horvitz 1990; Zilberstein 1996). Ideally, the solution to this problem is the optimal stopping point because it results in the optimum of the time-dependent utility function. However, the optimal stopping point of an anytime algorithm can be challenging to determine given substantial uncertainty over the performance of the anytime algorithm.

Generalized Meta-Level Control Problem

In this section, we propose a generalization of the standard meta-level control problem for anytime algorithms. Intuitively, an anytime algorithm can be viewed as containing an internal hyperparameter that either interrupts or executes the algorithm for another time step (i.e., *stopping*) along with a set of internal hyperparameters that adjust the internal operation of the algorithm (i.e., *hyperparameter tuning*). This results in a new meta-level control problem for both optimal stopping and hyperparameter tuning of an anytime algorithm. As we discuss later in the paper, our approach monitors and controls an anytime algorithm by expressing this new meta-level control problem as a deep RL problem. We present a description of an anytime algorithm below.

Definition 3. *An **anytime algorithm**, Λ, contains an internal hyperparameter $\Theta_0 = \{\text{STOP}, \text{CONTINUE}\}$ that either interrupts or executes the algorithm for another time step Δ along with a set of internal hyperparameters $\{\Theta_1, \ldots, \Theta_{\ell_\Theta}\}$ that adjusts the internal operation of the algorithm.*

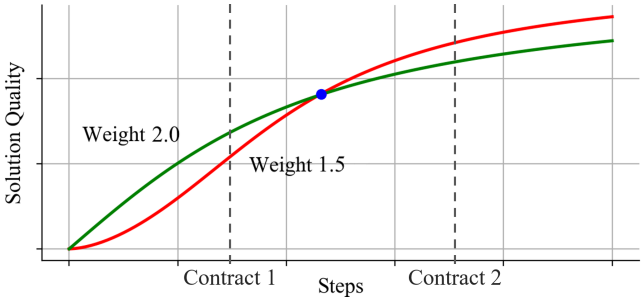

Figure 1: An illustration of anytime weighted A*.

*Anytime weighted A**, an anytime variant of the A* heuristic search algorithm (Hansen, Zilberstein, and Danilchenko 1997; Likhachev, Gordon, and Thrun 2004; Aine, Chakrabarti, and Kumar 2007; Hansen and Zhou 2007; Thayer and Ruml 2010) is a common example of an anytime algorithm. We use anytime weighted A* to illustrate our approach throughout the paper. The algorithm (1) uses an inadmissible heuristic to quickly find suboptimal solutions, (2) continues the search after each solution is found, (3) provides an error bound on each suboptimal solution, and (4) guarantees an optimal solution at termination. Notably, the standard evaluation function $f(n) = g(n) + h(n)$ used to select the next node for expansion from the open list is replaced with a weighted evaluation function $f_w(n) = g(n) + w \cdot h(n)$, where the path cost $g(n)$ is the cost of the path from the start node to a node n and the heuristic $h(n)$ is the estimated cost from a node n to the goal node. Intuitively, by weighting the heuristic $h(n)$ more heavily than the path cost $g(n)$ given any weight $w \geq 1$, the algorithm expands nodes that appear closer to reaching any solution instead of nodes that lead to the optimal solution, potentially speeding up the search at the expense of solution quality.

Figure 1 shows typical performance curves of two runs of anytime weighted A* with different weights that each solve a given instance of a problem. With *deadlines*, a weight of 2.0 leads to better quality at *Contract 1* while a weight of 1.5 results in better quality at *Contract 2*. Without *deadlines*, a weight of 2.0 leads to better quality in the short term but worse quality in the long term while a weight of 1.5 results in worse quality in the short term but better quality in the long term. This leads to an important question: is it possible to tune the hyperparameters of an anytime algorithm at runtime to optimize its performance *with* or *without* deadlines? We answer this question by offering a simple metareasoning framework for learning optimal stopping and hyperparameter tuning of anytime algorithms with deep RL.

Our metareasoning approach that uses deep RL represents the generalization of the standard meta-level control problem for anytime algorithms as an 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 a state $s' \in S$ after performing an action $a \in A$ in a state $s \in S$, $R : S \times A \times S \to \mathbb{R}$ represents the expected immediate reward of reaching a state $s' \in S$ after performing an action $a \in A$ in a state $s \in S$, and $s_0 \in S$ is a start state. A solution to an MDP is a policy $\pi : S \to A$ indicating that an action $\pi(s) \in A$ should be performed in a state $s \in S$. A policy π induces a value function $V^\pi : S \to \mathbb{R}$ representing the expected discounted cumulative reward $V^\pi(s) \in \mathbb{R}$ for each state $s \in S$ given a discount factor $0 \leq \gamma < 1$. An optimal policy π^* maximizes the expected discounted cumulative reward for every state $s \in S$ by satisfying the Bellman optimality equation $V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$.

Here, we express the generalization of the standard meta-level control problem for anytime algorithms as an MDP with a specific state and action space. The *states* have *state factors* that reflect the quality and computation time of the current solution along with *state factors* that reflect the internal state of the algorithm, the instance of the problem, or the performance of the underlying system. The *actions* have an *action factor* that reflect the internal hyperparameter that either interrupts or executes the algorithm for another time step along with *action factors* that reflect the internal hyperparameters that adjust internal operation of the algorithm. We formalize the generalized meta-level control problem for anytime algorithms as an MDP in the following way.

Definition 4. *The **generalized meta-level control problem** for monitoring and controlling an anytime algorithm, Λ, is represented by an MDP $\langle \Phi, \Psi, F, S, A, T, R, s_0 \rangle$ given a time-dependent utility function $U : \Phi \times \Psi \to \mathbb{R}$:*

- $\Phi = \{q_0, q_1, \ldots, q_{N_\Phi}\}$ *is a set of qualities.*
- $\Psi = \{t_0, t_1, \ldots, t_{N_\Psi}\}$ *is a set of time steps.*
- $F = F_0 \times F_1 \times \cdots \times F_{N_F}$ *is a set of features that summarize the internal state of the algorithm, the instance of the problem, or the performance of the underlying system.*
- $S = \Phi \times \Psi \times F$ *is a set of states of computation: each state $s \in S$ indicates that the algorithm has a solution of quality $q \in \Phi$ at time step $t \in \Psi$ with a feature $f \in F$.*
- $A = \Theta_0 \times \Theta_1 \times \cdots \times \Theta_{N_\Theta}$ *is a set of actions of computation: the internal hyperparameter $\Theta_0 = \{\text{STOP}, \text{CONTINUE}\}$ either interrupts or executes the algorithm for another time step while the internal hyperparameters $\Theta_1, \ldots, \Theta_{N_\Theta}$ adjust its internal operation.*
- $T : S \times A \times S \to [0, 1]$ *is an unknown, possibly nonstationary transition function that represents the probability of reaching a state $s' = (q', t', f') \in S$ after performing an action $a \in A$ in a state $s = (q, t, f) \in S$.*
- $R : S \times A \times S \to \mathbb{R}$ *is a reward function that represents the expected immediate reward, $R(s, a, s') = U(q', t') - U(q, t)$, of reaching a state $s' = (q', t', f') \in S$ after performing an action $a \in A$ in a state $s = (q, t, f) \in S$.*
- $s_0 \in S$ *is a start state $s_0 = (q_0, t_0, f_0) \in S$ that indicates that the algorithm has a solution of quality $q_0 \in \Phi$ at time step $t_0 \in \Psi$ with a feature $f_0 \in F$.*

Note that the reward function is consistent with the objective of optimizing the time-dependent utility: executing the anytime algorithm until a solution of quality $q \in \Phi$ at time step $t \in \Psi$ with a feature $f \in F$ gives a cumulative reward that is equal to the time-dependent utility $U(q, t)$. This is a form of *reward shaping*—equivalent to emitting a reward of $U(q, t)$

once at the end of an episode in terms of the objective—that accelerates reinforcement learning by guiding the agent with a reward at each time step (Ng, Harada, and Russell 1999).

Many metareasoning approaches to anytime algorithms use only the quality and computation time of the current solution as the state of computation (Zilberstein and Russell 1995). This state of computation, however, will likely not satisfy the *Markov* property. It could therefore benefit from features that summarize the internal state of algorithm, the instance of the problem, or the performance of the underlying system. As an example, in a domain that uses anytime weighted A* to solve an instance of a travelling salesman problem, there could be features for the mean of the g- and h-values on the open list of anytime weighted A*, the number of cities in the instance at hand (Hutter et al. 2014), or the processor usage of the system. Our metareasoning approach can naturally use a complex representation for the state of computation to approximate the Markov property by leveraging the advantages of using a neural network as a function approximation with reinforcement learning.

We show that an optimal policy for the meta-level control problem produces optimal meta-level control of an anytime algorithm under certain conditions below.

Remark 1. *If the change in the current solution of the anytime algorithm given a state of computation $s \in S$ and an action of computation $a \in A$ satisfies the Markov property, the optimal policy $\pi^* : S \to A$ results in optimal stopping and hyperparameter tuning.*

Proof Sketch. This follows directly from the Markov property: a transition to a successor state of computation $s' \in S$ only depends on the current state of computation $s \in S$ and the current action of computation $a \in A$. □

Meta-Level Control with Deep RL

Our meta-level control technique based on deep RL learns both optimal stopping and hyperparameter tuning for an anytime algorithm: it performs a series of episodes that each use the anytime algorithm to solve a generated instance of a specific problem. Generally, deep RL has been effective across a variety of applications, such as Atari (Mnih et al. 2015), chess (Silver et al. 2018), and StarCraft (Vinyals et al. 2019). A deep RL agent learns a policy as a neural network by performing actions and observing rewards in the world, making it a natural approach to the meta-level control problem for anytime algorithms for a few reasons. First, by balancing exploitation and exploration, it can learn how to adjust the algorithm's internal hyperparameters without knowing the transition function. Next, by ignoring large unreachable regions of the state space, it can reduce the overhead of learning how to adjust the algorithm's internal hyperparameters. Finally, by using a neural network that extracts the relationship between large input and output spaces, it can encode the effects of the algorithm's internal hyperparameters on its internal state in a way that generalizes to internal states that are unfamiliar or have not been seen yet.

Algorithm 1 shows our meta-level control technique that uses *deep Q-learning* (Mnih et al. 2015). Each episode (Line 4) starts by executing the anytime algorithm for a time step (of a duration Δ) on a generated instance of a specific problem (Lines 5-11). For each time step as the anytime algorithm is executing (Line 12), there are several steps. First, the experience buffer is updated with the current state of computation, the current action of computation, the current reward, and the next state of computation (Lines 13-15). Next, if the size of the experience buffer exceeds the initialization period, we sample a minibatch (Lines 16-17). With that minibatch, the temporal-difference error is used to update the action-value network via backpropagation and then the target action-value network is updated via a moving average (Lines 17-21). Finally, the anytime algorithm is either interrupted or executed for another time step while adjusting its internal hyperparameters by following the policy computed from the action-value network and the exploration strategy (Lines 22-28).

Algorithm 1: Our meta-level control technique.

Input: Anytime algorithm Λ, action-value network $\mathcal{N}$, step size α_1, target action-value network step size α_2, exploration strategy $\mathcal{E}$, experience buffer capacity ℓ_1, number of episodes ℓ_2, initialization period ℓ_3, minibatch size ℓ_4, and duration Δ
Output: An action-value function Q

1 $B \leftarrow$ ExperienceBuffer(ℓ_1)
2 $Q \leftarrow$ NeuralNetwork($\mathcal{N}$)
3 $\hat{Q} \leftarrow Q$
4 **for** $i = 1, 2, \ldots, \ell_2$ **do**
5 $\quad P \leftarrow$ SampleProblemDistribution()
6 $\quad \Lambda$.Setup(P)
7 $\quad t \leftarrow 0$
8 $\quad s_t \leftarrow (\Lambda.\text{Get}\Phi(), \Lambda.\text{Get}\Psi(), \Lambda.\text{Get}F())$
9 $\quad a_t \leftarrow \pi^Q_{\mathcal{E}}(s_t)$
10 $\quad \Lambda$.Start($a_t.\Theta_1, \ldots, a_t.\Theta_{\ell_\Theta}$)
11 $\quad$ Sleep(Δ)
12 $\quad$ **while** Λ.Running() **do**
13 $\quad\quad s_{t+1} \leftarrow (\Lambda.\text{Get}\Phi(), \Lambda.\text{Get}\Psi(), \Lambda.\text{Get}F())$
14 $\quad\quad r_t \leftarrow R(s_t, a_t, s_{t+1})$
15 $\quad\quad B$.Append((s_t, a_t, r_t, s_{t+1}))
16 $\quad\quad$ **if** B.Size() $\geq \ell_3$ **then**
17 $\quad\quad\quad M \leftarrow B$.SampleMinibatch($\ell_4$)
18 $\quad\quad\quad \hat{\mathcal{L}}(r, s') \coloneqq r + \gamma \max_{a' \in A} \hat{Q}(s', a')$
19 $\quad\quad\quad \mathcal{L}(s, a, r, s') \coloneqq (\hat{\mathcal{L}}(r, s') - Q(s, a))^2$
20 $\quad\quad\quad Q$.Backpropagate($M, \mathcal{L}, \alpha_1$)
21 $\quad\quad\quad \hat{Q} \leftarrow (1 - \alpha_2) \cdot \hat{Q} + \alpha_2 \cdot Q$
22 $\quad\quad t \leftarrow t + 1$
23 $\quad\quad a_t \leftarrow \pi^Q_{\mathcal{E}}(s_t)$
24 $\quad\quad$ **if** $a_t.\Theta_0 =$ Stop **then**
25 $\quad\quad\quad \Lambda$.Stop()
26 $\quad\quad\quad$ **break**
27 $\quad\quad \Lambda$.Continue($a_t.\Theta_1, \ldots, a_t.\Theta_{\ell_\Theta}$)
28 $\quad\quad$ Sleep(Δ)
29 $\quad$ **return** Q

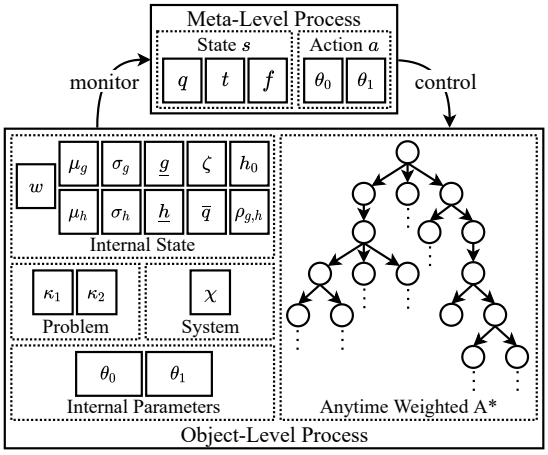

Figure 2: An example of a metareasoning architecture.

Anytime Weighted A* Example

We now apply our approach to anytime weighted A*. Recent work on anytime weighted A* focuses on selecting the best static weight for a problem (Hansen and Zhou 2007), choosing the best static weight for an instance of a problem (Sun, Druzdzel, and Yuan 2007), and changing the weight at runtime heuristically (Thayer and Ruml 2009). There is also work that shows that anytime weighted A* can be improved via restarting when a solution is found (Richter, Thayer, and Ruml 2010) and work that analyzes the failure conditions of anytime weighted A* with respect to its weight (Wilt and Ruml 2012). Overall, recent work highlights the difficulty of adjusting the weight of anytime weighted A* at runtime.

The meta-level control problem for anytime weighted A*, Λ, is an MDP $\langle \Phi, \Psi, F, S, A, T, R, s_0 \rangle$. $\Phi = [0, 1]$ is the set of qualities. $\Psi = [0, \tau]$ is the set of time steps with a deadline τ. F is the set of features such that the feature $w \in W$ is the current weight, the features $\mu_g \in \mathbb{R}$ and $\mu_h \in \mathbb{R}$ are the mean of the g- and h-values on the open list, the features $\sigma_g \in \mathbb{R}$ and $\sigma_h \in \mathbb{R}$ are the standard deviation of the g- and h-values on the open list, the features $\underline{g} \in \mathbb{R}$ and $\underline{h} \in \mathbb{R}$ are the minimum g- and h-values on the open list, the feature $\zeta \in \mathbb{R}$ is the value $\log(n)$ for the number of nodes n on the open list, the feature $\bar{q} \in \mathbb{R}$ is the h-value of the initial state divided by the minimum f-value on the open list, the feature h_0 is the h-value of the initial state, the feature $\rho_{gh} \in [-1, 1]$ is the correlation between the g- and h-values on the open list, the feature $\kappa \in K$ is the settings for the instance of the problem, and the feature $\chi \in [0, 1]$ is the processor usage of the system. A is the set of actions: $\Theta_0 \in \{\text{STOP}, \text{CONTINUE}\}$ interrupts or executes the algorithm for another time step Δ while $\Theta_1 = \{\leftarrow, \rightarrow\}$ adjusts the weight $w \in W$ gradually by shifting its pointer to the set of weights $W = \{1, 1.5, 2, 3, 4, 5\}$ left or right, which is sufficient when the state transitions are also gradual. Note that S, A, T, R, and s_0 follow directly from the generalized meta-level control problem for anytime algorithms.

Similar to recent work on metareasoning (Svegliato et al. 2019, 2022), the metareasoning architecture in Figure 2 has a *meta-level process* that monitors and controls an *object-level process* that executes anytime weighted A*.

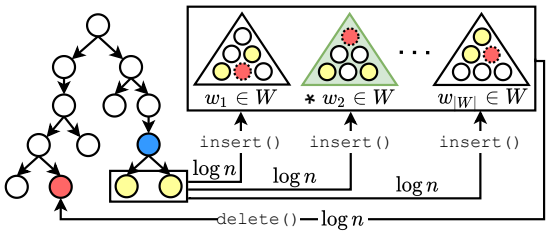

Figure 3: A modified implementation of anytime weighted A* that manages multiple open lists for different weights.

Anytime weighted A* involves a simple modification to allow its weight to be adjusted at runtime. Instead of inserting/deleting a node into/from a single open list for a static weight, the algorithm inserts/deletes this node into/from $|W|$ virtual open lists each ordered by the f_w-value of a weight $w \in W$ as shown in Figure 3 such that each open list has a different ordering of the same exact nodes. This leads to a worst-case time complexity of $\mathcal{O}(|W| \log n)$ for sequentially inserting/deleting a node across all $|W|$ virtual open lists of size n. In practice, thanks to the efficient implementation of anytime weighted A* used throughout our experiments, we confirmed that the overhead of managing multiple virtual open lists is negligible (less than 1%).

Experiments

We now evaluate our approach on a common benchmark domain that uses anytime weighted A* to solve a range of heuristic search problems and a mobile robot application that uses RRT* to solve motion planning problems.[1]

In our experiments, our approach and the standard approaches each solve 1000 random test instances of a given problem using the anytime algorithm from both domains. For each random test instance, we continuously measure the solution quality produced by the anytime algorithm across all approaches. A solution quality $q = 0$ means no solution was computed while a solution quality $q = 1$ means an optimal solution was computed. Formally, for any instance of a problem, the quality of a given solution is ideally defined as the approximation ratio, $q = c^*/c$, where c^* is the cost of the optimal solution and c is the cost of the given solution. However, since computing the optimal solutions for the complex problems in our experiments is often infeasible, we estimate the quality of a given solution as the approximation ratio, $q = \hat{c}^*/c$, where $\hat{c}^*$ is a lower bound on the cost of the optimal solution and c is the cost of the given solution, following earlier work (Hansen and Zilberstein 2001).

Naturally, any meta-level control problem for anytime algorithms requires a time-dependent utility function. In our experiments, we consider a common *stopping-deadline* setting: there is an increasing urgency for a solution to incentivize early termination of the anytime algorithm along with

[1] A public Julia library offers the implementation of our approach and the RL environments for anytime weighted A* and RRT*: ⟨https://github.com/bhatiaabhinav/Metareasoning.jl⟩.

a mandatory deadline of τ seconds that incurs a severe deadline penalty Υ. This is common in planning and robotics where an autonomous system urgently needs a solution but also needs that solution prior to a mandatory deadline. More formally, given a solution of quality $q \in \Phi$ at time step $t \in \Psi$, the time-dependent utility function is as follows:

$$U(q,t) = [t \leq \tau] \cdot (U_I(q) - U_C(t)) - [t > \tau] \cdot \Upsilon,$$

where $U_I(q) = \iota q$ is the intrinsic value function and $U_C(t) = e^{\beta t} - 1$ is the cost of time given the scalar ι and β selected in practice based on the value of a solution and the urgency for a solution (Hansen and Zilberstein 2001). We want to highlight that our approach generally supports any well-behaved time-dependent utility function in addition to the time-dependent utility function considered in this paper.

Our approach is trained on random training instances for each problem in the domains. These problems only involve a few hours of training. Algorithm 1 uses typical settings for deep Q-learning. The action-value network $\mathcal{N}$ is a fully connected neural network with two hidden layers of 64 and 32 nodes with ReLU activation and a linear output layer of 6 nodes. The step size α_1 is 0.0001. The target action-value network step size α_2 is 0.001. The exploration strategy $\mathcal{E}$ is ϵ-greedy action selection with an exploration probability ϵ that is annealed from 1 to 0.1 over 1000 episodes. The experience buffer capacity ℓ_1 is ∞. The number of episodes ℓ_2 is 15000 and 30000 for the anytime weighted A* and RRT* domains. The initialization period ℓ_3 is 1000. The minibatch size ℓ_4 is 128. The duration Δ is $1/20$th of the deadline. Our experiments were run on an AMD Ryzen 3900X CPU with 32 GB of RAM using a fixed set of random seeds. Note that the random training instances for each problem differ from the random test instances to ensure that our approach generalizes to unseen or unfamiliar instances of each problem.

Common Benchmark Domain

We first experiment with a common benchmark domain that uses anytime weighted A* to solve a range of heuristic search problems by comparing our approach to the standard approaches. For our approach, we consider two variants: DQN$_\tau$ can adjust the weight but always executes anytime weighted A* until the deadline τ while DQN(t) can both adjust the weight and interrupt anytime weighted A* at an earlier stopping point t. The standard approaches always execute anytime weighted A* until the deadline τ and use either a static weight (denoted by a *number*) or a dynamic weight that decreases from the highest weight after each solution (denoted by DEC), given a set of common weights of 1, 1.5, 2, 3, 4, and 5 (Richter, Thayer, and Ruml 2010). Overall, as we discuss later, our results demonstrate that our approach outperforms the standard approaches regardless of whether we always execute anytime weighted A* until the deadline (DQN$_\tau$) or can interrupt anytime weighted A* at an earlier stopping point (DQN(t)).

We provide a description of the heuristic search problems below. Each problem is selected to reflect problems that require different static weights and problems for which counterintuitive behavior of anytime weighted A* was reported (Wilt and Ruml 2012). The parameters of each problem are selected to avoid trivializing the problem by either not having enough time so that no approach finds any solution or having too much time so that every approach finds the optimal solution before the deadline. For the time-dependent utility function, the scalars ι and β are 1 and $\ln(1.25)$, the deadline penalty Υ is roughly ∞, and the deadline is $\tau = 1$ second corresponding to roughly 100000, 100000, 50000, and 500000 node expansions for the SP, ISP, TSP, and GNP benchmark problems on our system using our implementation. This means that there is a cost of time $U_C(t)$ and a mandatory deadline τ. Note that we enforce the node expansion limit instead of the deadline for reproducibility.

Sliding Puzzle An SP instance has $J = j^2 - 1$ tiles with each tile i labeled from 1 to J in a $j \times j$ grid. Every tile must be moved from an initial position to a desired position given a *unit* cost $c(i) = 1$ for moving a tile i. The sum of the Manhattan distances from the current position of each tile to its desired position is used as an admissible and consistent heuristic function h. The number of tiles J is 15. The difficulty of an instance, as measured by the h-value of the initial configuration of all tiles, is chosen randomly between 35 and 45. The MDP includes the setting K that represents the difficulty for the instance of the problem.

Inverse Sliding Puzzle An ISP instance is the same as an SP instance except that there is an *inverse* cost $c(i) = 1/i$ for moving a tile i. This means that the sum of the Manhattan distances from the current position of each tile to its desired position, weighted by the cost of moving each tile, is used as an admissible and consistent heuristic function h.

Traveling Salesman Problem A TSP instance has J cities that must be visited along an optimal route given a cost for each edge between a pair of cities. A percentage of the edges have an infinite cost to control its sparsity. The total cost of a minimum spanning tree across the unvisited cities with an infinite cost for no feasible tour is used as an admissible and consistent heuristic function h. The number of cities J is chosen randomly between 25 and 35. The percentage of edges with an infinite cost is chosen randomly between 0% and 90%. The cost for each edge between a pair of cities is chosen randomly $\in (0, 1]$. The MDP includes the settings K that represent the number of cities and the percentage of edges with an infinite cost for the problem instance.

Grid Navigation Problem A GNP instance has a grid of $j \times j$ cells with a percentage of the cells marked as obstacles. The obstacles of shape $1 \times i$ are scattered randomly throughout the grid. The goal is to find the shortest path from an initial position of $(1, 1)$ to a goal position (j, j) by moving north, east, south, or west, which incurs unit cost. The Manhattan distance from the current position to the goal position is used as an admissible and consistent heuristic function h. The grid is set to $j = 1000$ and $i = 10$. The density of the obstacles is chosen randomly between 5% and 10%. The MDP includes the setting K for the density of the obstacles.

Common Benchmark Results Figure 4 demonstrates that our DQN$_\tau$ approach outperforms the best standard approach on each benchmark problem: it exhibits a better mean time-

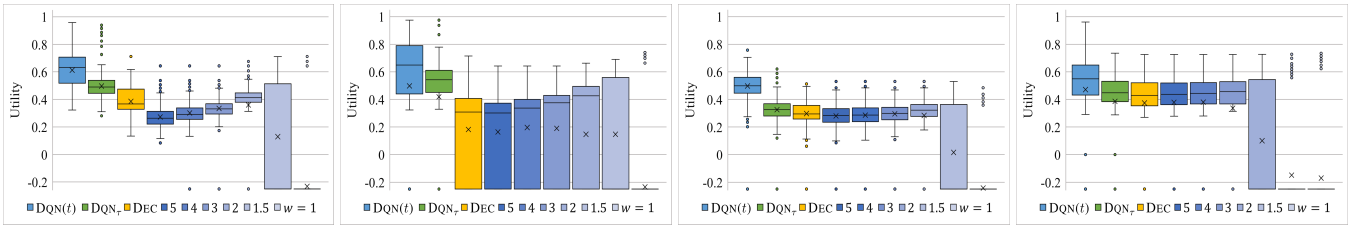

Figure 4: The box plots of the final time-dependent utilities produced by anytime weighted A* for each approach over all instances of SP, ISP, TSP, and GNP (from *left* to *right*). Note that the crosses represent the mean, the bullets represent the outliers, and the time-dependent utilities can be negative due to the cost of time $U_C(t)$ and the deadline penalty Υ.

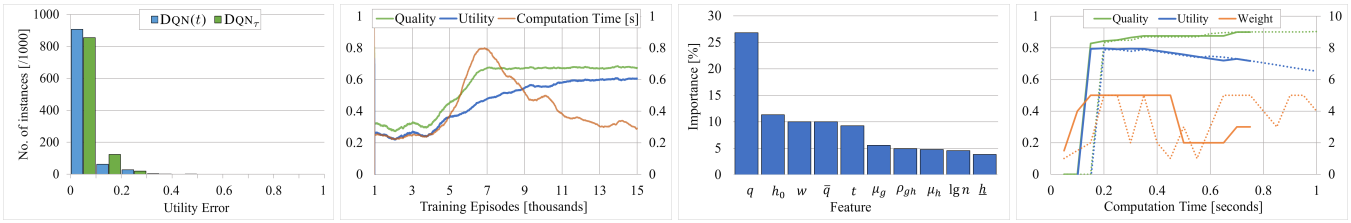

Figure 5: An analysis of our approach (from *left* to *right*). (a) For SP, a histogram showing the distribution of every instance over the *utility error* for each of our approaches. We define *utility error* as the normalized difference between the final time-dependent utility of each of our approaches and the final time-dependent utility of the best approach that always executes anytime weighted A* until the deadline. (b) For SP, a smoothed line chart showing the final solution quality (*left y-axis*), the final time-dependent utility (*left y-axis*), and the final computation time (*right y-axis*) for each training episode of our DQN(t) approach. (c) For SP, a bar chart showing the *importance* of the top 10 features used by our DQN(t) approach. We define *importance* as the normalized mean absolute weight on a feature in the input layer of the neural network of our DQN(t) approach. (d) On a selected instance of GNP, a line chart showing how the solution quality (*left y-axis*), the time-dependent utility (*left y-axis*), and the weight of anytime weighted A* (*right y-axis*) vary with the computation time as anytime weighted A* solves the select instance using our DQN$_\tau$ approach (*dotted lines*) and our DQN(t) approach (*solid lines*). We emphasize that our DQN$_\tau$ approach executes anytime weighted A* until the deadline while our DQN(t) approach interrupts anytime weighted A* at the earlier stopping point.

dependent utility than DEC for SP and $w = 4$ for ISP and a comparable time-dependent utility to DEC for TSP and $w = 5$ for GNP. Hence, we observe that our DQN$_\tau$ approach is better than or comparable to the best standard approaches without any need to adjust the weight manually for each benchmark problem. More importantly, once we introduce stopping on top of adjusting the weight automatically, our DQN(t) approach further outperforms the best standard approach on each benchmark problem. This highlights that our deep RL metareasoning approach for anytime algorithms can learn both stopping and hyperparameter tuning unlike existing methods that only optimize for one or the other.

Figure 5 offers an analysis of our approach. Figure 5(a) shows that our DQN(t) and DQN$_\tau$ approaches exhibit a utility error of 0 for about 900 and 850 instances (out of 1000) of the SP benchmark problem. This means it leads to the highest time-dependent utility more often than the standard approaches, which explains our results in Figure 4. Figure 5(b) shows that, as our DQN(t) approach learns to optimize the time-dependent utility for the SP benchmark problem, it initially focuses on improving the quality of the solution and later focuses on reducing the computation time by terminating early. Hence, at the end of training, the agent learns how to balance the quality of the solution and the computation time. Figure 5(c) shows that the decisions of our DQN(t) approach are informed by key features like the quality q of the current solution, the heuristic value h_0 of the starting state, the current weight w of anytime weighted A*, the upper bound $\bar{q}$ on the solution quality, and the computation time t. Figure 5(d) shows a selected instance of the GNP benchmark problem as it solved by our DQN$_\tau$ approach and our DQN(t) approach: the former terminates with a solution of slightly higher quality but lower time-dependent utility while the latter terminates with a solution of slightly lower quality but higher time-dependent utility. Thus, we observe that interrupting anytime weighted A* at an earlier stopping point once its performance diminishes is beneficial.

Mobile Robot Application

We now experiment with the mobile robot application that uses RRT* to solve motion planning problems by comparing our approach to a standard approach. RRT* is a popular algorithm that converges in the limit to an optimal motion plan from a start state to a goal state by rapidly expanding a tree via sampling the map randomly (Karaman et al. 2011). Typically, RRT* has two hyperparameters: a growth factor limiting how much the tree grows for each sample and an area of focus biasing where each sample is drawn. Recent work on RRT* focuses on heuristically tuning the growth factor and area of focus (Urmson and Simmons 2003; Ak-

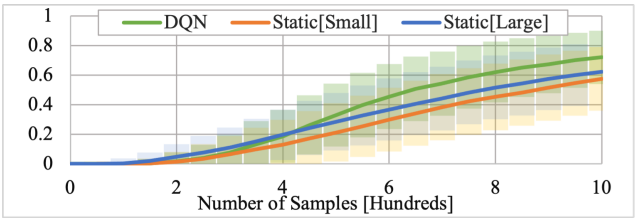

Figure 6: The performance curves showing how the mean solution quality varies with the number of samples for all instances of the motion planning problem.

gun and Stilman 2011; Kiesel, Burns, and Ruml 2012).

In particular, we compare our approach to a standard approach that executes RRT* with a small and large static growth factor and an area of focus that spans the entire map. Each motion planning problem is a map with a high density of obstacles where the start and goal states are in the bottom-left and top-right corners. For the time-dependent utility function, the scalars ι and β are 1 and 0, the deadline penalty is roughly ∞, and the deadline τ corresponds to 1000 samples. This means that there is a mandatory deadline τ but no cost of time $U_C(t)$. Note that we again enforce a sample limit instead of the deadline for reproducibility.

The meta-level control problem for RRT* is similar to anytime weighted A* but with a distinct set of features and actions of computation. The features include the growth factor, the position of the area of focus, the percentage of samples that have expanded the tree, a lower bound on the remaining distance to the goal state, an estimate of the free space of the map, an estimated average size of the obstacle, and a score for each possible area of focus that considers an estimated probability of improving the current path, the fraction of the current path within that area of focus, the fraction of the tree within that area of focus, and the average curvature of the current path within that area of focus. The actions of computation include the internal hyperparameter that increases or decreases the growth factor and the internal hyperparameter that moves the position of the area of focus.

Mobile Robot Results Figure 6 shows that our approach outperforms the standard approach to RRT*. At the maximum number of samples, our approach produces a higher mean solution quality than the standard approach by adjusting the growth factor and the area of focus. In particular, our approach achieves a mean solution quality of roughly 0.73 while the standard approach to RRT* achieves a mean solution quality of 0.62 and 0.59 for the large and small growth factor. Moreover, over any number over 450 samples, our approach is comparable to or better than the standard approach to RRT*. This indicates that it is sampling in a more efficient way that results in better motion plans. Overall, we find these results especially encouraging because our deep RL architecture that was originally configured for anytime weighted A* did not need to be updated for RRT*.

Figure 7 illustrates qualitatively how our approach adjusts RRT* on selected instance. In general, our approach guides the tree from the start state to the goal state of the map by

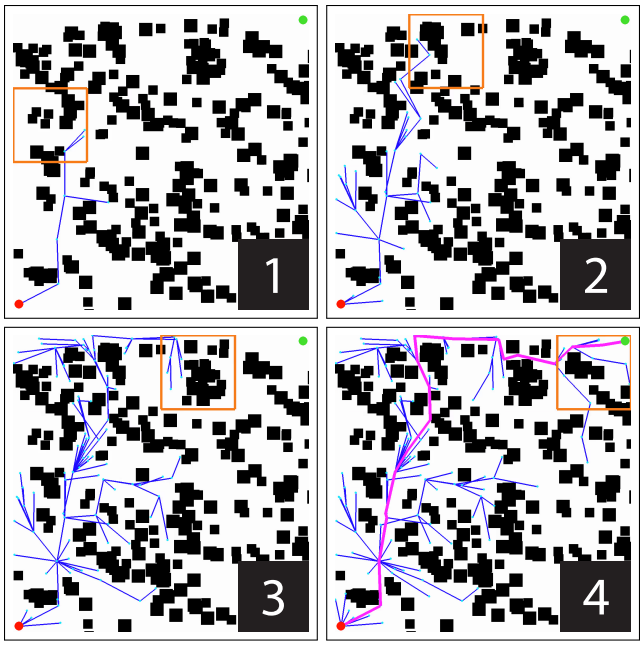

Figure 7: The evolution of RRT* over the number of samples for our approach on a selected instance of the motion planning problem from Checkpoint 1 to 4. The *black* shapes are the obstacles, the *green* and *red* circles are the start and goal states, the *blue* lines are the current tree, the *cyan* circles are the samples, the *purple* line is the current path, and the *orange* box is the current area of focus.

focusing on the frontier that has been less explored to compute an initial path quickly. That is, from Checkpoint 1 to 4, the area of focus is typically placed at the frontier of the existing tree of RRT*. Intuitively, our approach shifts the area of focus to the frontier to compute an initial motion plan as quickly as possible (*shown*). Thereafter, once our approach has computed an initial motion plan, it favors the center of the map near the initial motion plan to improve it as rapidly as possible (*not shown*). Overall, our approach learns how to shift the area of focus to key areas of the map, outperforming the standard approach that samples from the entire map, by reducing how long it takes to compute an initial motion plan and how long it takes to improve that initial motion plan.

Conclusion

We present a general, decision-theoretic metareasoning approach that optimizes both the stopping point and hyperparameters of anytime planning. It not only boosts the performance of an anytime algorithm by tuning its hyperparameters at runtime on a specific instance of a problem, but also eliminates any need for manual hyperparameter exploration. Most importantly, we show that our meta-level control technique learns how to monitor and control anytime algorithms that reflect distinct problem solving paradigms—heuristic search and motion planning—without changing any of the deep RL settings. Future work will explore additional anytime algorithms commonly used in planning and robotics.

Acknowledgments

This work was supported by the NSF GRFP DGE-1451512 and the NSF grants IIS-1813490 and IIS-1954782.

References

Adenso-Diaz, B.; and Laguna, M. 2006. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations research*, 54(1).

Aine, S.; Chakrabarti, P.; and Kumar, R. 2007. AWA*: A window constrained anytime heuristic search algorithm. In *20th IJCAI*.

Akgun, B.; and Stilman, M. 2011. Sampling heuristics for optimal motion planning in high dimensions. In *IROS*.

Ansótegui, C.; Sellmann, M.; and Tierney, K. 2009. A gender-based genetic algorithm for the automatic configuration of algorithms. In *15th CP*.

Bhatia, A.; Svegliato, J.; and Zilberstein, S. 2021. On the benefits of randomly adjusting anytime weighted A*. In *12th SOCS*.

Biedenkapp, A.; Bozkurt, H. F.; Eimer, T.; Hutter, F.; and Lindauer, M. 2020. Dynamic algorithm configuration: Foundation of a new meta-algorithmic framework. In *24th ECAI*.

Birattari, M.; Stützle, T.; Paquete, L.; and Varrentrapp, K. 2002. A racing algorithm for configuring metaheuristics. In *4th GECCO*.

Birattari, M.; Yuan, Z.; Balaprakash, P.; and Stützle, T. 2010. F-Race and iterated F-Race: An overview. *Experimental Methods for the Analysis of Optimization Algorithms*.

Boddy, M.; and Dean, T. L. 1994. Deliberation scheduling for problem solving in time-constrained environments. *AI*, 67(2).

Burns, E.; Ruml, W.; and Do, M. B. 2013. Heuristic search when time matters. *JAIR*, 47.

Hansen, E. A.; and Zhou, R. 2007. Anytime heuristic search. *JAIR*, 28.

Hansen, E. A.; and Zilberstein, S. 2001. Monitoring and control of anytime algorithms: A dynamic programming approach. *AI*, 126.

Hansen, E. A.; Zilberstein, S.; and Danilchenko, V. A. 1997. Anytime Heuristic Search: First Results. Technical Report 97-50, Computer Science Department, University of Massachusetts Amherst.

Horvitz, E. 1988. Reasoning under varying and uncertain resource constraints. In *7th AAAI*.

Horvitz, E.; and Rutledge, G. 1991. Time-dependent utility and action under uncertainty. In *7th UAI*.

Horvitz, E. J. 1987. Reasoning about beliefs and actions under computational resource constraints. In *3rd UAI*.

Horvitz, E. J. 1990. *Computation and action under bounded resources*. Ph.D. thesis, Stanford University, CA.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Murphy, K. 2010. Time-bounded sequential parameter optimization. In *4th LION*.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Murphy, K. P. 2009a. An experimental investigation of model-based parameter optimisation: SPO and beyond. In *11th GECCO*.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009b. ParamILS: An automatic algorithm configuration framework. *JAIR*, 36.

Hutter, F.; Hoos, H. H.; and Stützle, T. 2007. Automatic algorithm configuration based on local search. In *22nd AAAI*.

Hutter, F.; Hoos, H. H.; et al. 2011. Sequential model-based optimization for general algorithm configuration. In *5th LION*.

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *AI*, 206.

Karaman, S.; Walter, M. R.; Perez, A.; Frazzoli, E.; and Teller, S. 2011. Anytime motion planning using the RRT*. In *ICRA*.

Karayev, S.; Fritz, M.; and Darrell, T. 2014. Anytime recognition of objects and scenes. In *27th CVPR*.

Kiesel, S.; Burns, E.; and Ruml, W. 2012. Abstraction-guided sampling for motion planning. In *5th SOCS*.

Likhachev, M.; Gordon, G.; and Thrun, S. 2004. ARA*: Anytime A* with provable bounds on sub-optimality. In *17th NeurIPS*.

Lin, C. H.; Kolobov, A.; Kamar, E.; and Horvitz, E. 2015. Metareasoning for planning under uncertainty. In *24th IJCAI*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540).

Ng, A. Y.; Harada, D.; and Russell, S. J. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *16th ICML*.

Paul, C. J.; Acharya, A.; Black, B.; and Strosnider, J. K. 1991. Reducing problem-solving variance to improve predictability. *Communications of the ACM*, 34(8).

Ramos, F. T.; and Cozman, F. G. 2005. Anytime anyspace probabilistic inference. *IJAR*, 38(1).

Richter, S.; Thayer, J. T.; and Ruml, W. 2010. The joy of forgetting: Faster anytime search via restarting. In *20th ICAPS*.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419).

Spaan, M. T.; and Vlassis, N. 2005. Perseus: Randomized point-based value iteration for POMDPs. *JAIR*, 24.

Sun, X.; Druzdzel, M. J.; and Yuan, C. 2007. Dynamic weighting A* search-based MAP algorithm for Bayesian networks. In *20th IJCAI*.

Svegliato, J.; Basich, C.; Saisubramanian, S.; and Zilberstein, S. 2022. Metareasoning for safe decision making in autonomous systems. In *ICRA*.

Svegliato, J.; Sharma, P.; and Zilberstein, S. 2020. A model-free approach to meta-level control of anytime algorithms. In *ICRA*.

Svegliato, J.; Wray, K. H.; Witwicki, S. J.; Biswas, J.; and Zilberstein, S. 2019. Belief space metareasoning for exception recovery. In *IROS*.

Svegliato, J.; Wray, K. H.; and Zilberstein, S. 2018. Meta-level control of anytime algorithms with online performance prediction. In *27th IJCAI*.

Svegliato, J.; and Zilberstein, S. 2018. Adaptive metareasoning for bounded rational agents. In *1st Workshop on AEGAP*.

Thayer, J.; and Ruml, W. 2009. Using distance estimates in heuristic search. In *19th ICAPS*.

Thayer, J.; and Ruml, W. 2010. Anytime heuristic search: Frameworks and algorithms. In *3rd SOCS*.

Urmson, C.; and Simmons, R. 2003. Approaches for heuristically biasing RRT growth. In *IROS*.

Vinyals, O.; Babuschkin, I.; Czarnecki, W. M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D. H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782).

Wilt, C.; and Ruml, W. 2012. When does weighted A* fail? In *5th SOCS*.

Zilberstein, S. 1996. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3).

Zilberstein, S.; and Russell, S. J. 1995. Approximate reasoning using anytime algorithms. In Natarajan, S., ed., *Imprecise and Approximate Computation*. Springer.