Learning When to Quit: Meta-Reasoning for Motion Planning

Yoonchang Sung, Leslie Pack Kaelbling, and Tomás Lozano-Pérez

***Abstract*— Anytime motion planners are widely used in robotics. However, the relationship between their solution quality and computation time is not well understood, and thus, determining when to quit planning and start execution is unclear. In this paper, we address the problem of deciding when to stop deliberation under bounded computational capacity, so called *meta-reasoning*, for anytime motion planning. We propose data-driven learning methods, model-based and model-free meta-reasoning, that are applicable to different environment distributions and agnostic to the choice of anytime motion planners. As a part of the framework, we design a convolutional neural network-based optimal solution predictor that predicts the optimal path length from a given 2D workspace image. We empirically evaluate the performance of the proposed methods in simulation in comparison with baselines.**

I. Introduction

Anytime algorithms [1] are a class of algorithms that improve solution quality over time and output an answer even if interrupted during computation. Such algorithms occur frequently in time-critical robotic applications and they raise decision-theoretic problems whose objective is to determine when to stop the computation, that is, stop improving the solution, and take action. In this work, we specifically focus on a class of anytime *motion planning* algorithms [2]–[4]. The goal of motion planning (in Figure 1) is to find a collision-free path from the initial configuration to the goal configuration such that the robot does not collide with any obstacles. Anytime motion planning algorithms typically improve the total length of the path when allowed additional computation time, although one could instead improve another objective, such as obstacle clearance or energy consumption.

Anytime motion planning consist of two phases: (1) finding a feasible path that reaches the goal configuration, and then (2) monotonically improving the path over time. We focus on the second phase in this work: we consider the problem of determining when to stop deliberation after finding a first feasible path. For anytime motion planning, we do not typically have a well-characterized relationship between the solution quality and computation time. Even for asymptotically optimal sampling-based motion planners there does not yet exist a theoretical understanding of their convergence rate to optimality [5]. Therefore, we will have to rely on data-driven methods to decide when to terminate the computation.

Meta-reasoning [6]–[8] (also known as meta-level control) is reasoning about reasoning, to address decision-theoretic deliberation under bounded computational capacity. Meta-reasoning can be used to determine when to stop deliberation by monitoring the progress of reasoning and regulating the computation time [9]. Our objective is to design a meta-reasoning framework for anytime motion planning that applies to any anytime motion planner without modification and generalizes to different environment distributions.

The authors are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA {yooncs8, lpk,tlp}@csail.mit.edu.

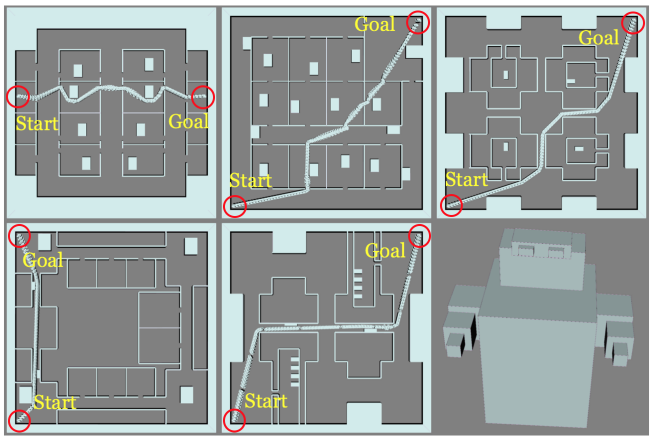


Fig. 1. Examples of environment distributions and the robot shape used in simulation.

Our meta-reasoning methods make decisions based on the observed profile of the quality of the current best solution over time (called the *performance profile*). If this profile is fairly smooth, meta-reasoning can be solved by online prediction without the necessity of data from previous runs; one can extrapolate the future solution quality from a model regressed from the history of solution qualities over the profile at hand [10]. This is, however, infeasible in anytime motion planning because performance profiles are generally highly non-smooth due to homotopy-class changes occurring stochastically (see Figure 2). Therefore, we will pursue data-driven learning methods to leverage meta-reasoning based data from related planning problems.

We assume the existence of a distribution of problems of interest, which share some similarity in their performance profiles, and assume examples from this distribution are available for training a model that can be used for efficient execution on new instances from that distribution. We generate training data by running a given anytime motion planner on the training problems and extracting the resulting performance profiles. From this data, we learn a decision policy by either model-based or model-free strategies. At execution time, we apply the learned policy to the evolving performance profile of a new problem, assuming that the given problem is drawn from the same distribution as the

training problems. Our methods can exploit aspects of the history of solution qualities to determine whether to stop deliberation or keep executing at every decision-making time.

Our model-based approach follows [11], using a discretization of planning time and solution quality to frame the decision problem as a Markov decision process (MDP). We learn a transition model for this discrete state and action MDP based on the historical performance profiles in the dataset. We can then compute an optimal stopping policy for this MDP using value iteration. This simple strategy is effective for small state spaces but does not scale well (in computation and data requirements) as the size of the state space increases, for example, when extending states to include additional features of the history besides the current quality and time.

We also investigate model-free approaches that use the function-approximation capabilities of neural networks to mitigate the curse of dimensionality. As meta-reasoning is a control problem, approximate dynamic programming or reinforcement learning (RL) can be adopted to learn a policy [12]. However, we observe that in our problem, we have access to an oracle for the optimal decision policy for each performance profile in the dataset, simply by letting the motion planner run long enough so that we get diminishing returns and determining, *post hoc*, the optimal stopping time for each training example. Therefore, model-free meta-reasoning can be formulated as *imitation learning* [13], where the optimal decision policy is treated as an expert demonstration. By exploiting this property, we can take a supervised learning approach, which is generally more robust and has lower sample complexity than RL. We propose two supervised learning-based methods: (1) neural network-based classification with hand-designed features and (2) recurrent neural network (RNN) sequence-to-sequence mapping on the raw performance profile.

In order to generalize over problem instances of different difficulty, the performance profiles need to be normalized so that the states, including quality and time, are comparable across problems. A critical requirement for this normalization is an estimate of the optimal path length for the current problem. We have designed a predictor for the optimal solution using a convolutional neural network (CNN) to predict a shortest path length for a given 2D workspace image. We use the predicted optimal solution as a normalizer. Although this approach could potentially be extended to 3D workspaces using voxel grids or point clouds as input, it currently limits the applicability of our methods to problems with planar workspaces.

Our contributions include:

- the application of machine-learning methods for general-purpose meta-reasoning to the problem of trading off planning and execution time for anytime motion planning;
- the formulation, implementation, and comparison of model-based and model-free strategies for this problem; and
- the use of CNNs to predict optimal solution quality from 2D images. We develop a solution predictor that predicts an optimal path length from a 2D workspace image.

We present experimental results demonstrating the effectiveness of these techniques in comparison to a set of baseline strategies.

II. Problem Description

Given an anytime motion planning algorithm and a dataset of performance profiles from previous motion planning problems, we want to find a policy π that can be used to decide when computation should be halted on a new problem. The goal is to receive the maximal expected *utility*, defined as a function of path quality and computation time. In this paper, we assume that we have a geometric and kinematic description of the robot and that the robot is operating in a 2D workspace, but that the dimension of the configuration space of the robot is unconstrained.

The anytime motion planner, after having found an initial path, can be thought of as a function that produces a path quality value as a function of running time; this function is monotonically non-decreasing. We will normalize this function, as described later, so that the running time t is in the range $[0, 1]$ and the quality q is also in the range $[0, 1]$. We are also given a normalized utility $U(q, t)$ that is a function of the normalized quality q and normalized running time t. We will assume that there are $T+1$ evenly spaced decision steps, $i \in [0, T]$. At every time step i, the performance history H_i is a vector of all the normalized utility values up to that time step. The robot picks an action $a_i = \pi(H_i)$ at each decision time step i in the current, unseen, workspace. If the robot chooses a_i to be `continue`, the deliberation continues to the $(i+1)$-th time step and repeats the same process at time t_{i+1}. The meta-reasoning process terminates if $a_i =$ `stop` or $i = T$.

We seek a utility-maximizing stopping policy $\pi(H_i)$. An ideal policy with oracular information would output $a_i =$ `stop` for $i = \arg\max_{j \in [0,T]} U(q_j, t_j)$. We can at best seek a policy that is optimal in expectation given the performance profile so far, which is to stop if we do not expect any future time to have a higher utility than the current one:

$$\pi(H_i) = \texttt{continue} \quad \text{iff} \\ \exists j \in [i+1, T],\ \mathbb{E}\big[U(q_j, t_j) \mid H_i\big] > U(q_i, t_i) \ .$$

Let the time at which the first feasible solution is found be, without loss of generality, 0. The *running time* is the maximal allowed time for attempting to improve the solution, at which time meta-reasoning always terminates the algorithm. We define a normalized time t as:

$$t = \frac{\text{current time}}{\text{running time}} \in [0, 1]. \tag{1}$$

We discretize the normalized time range into $T + 1$ values.

The solution quality, denoted by q, will also be normalized. We use the optimal (*i.e.*, shortest) path length for this problem and the worst (*i.e.*, longest) path length for this

problem for normalization, so that the normalized solution quality q is:

$$q = \frac{\text{worst path length} - \text{current path length}}{\text{worst path length} - \text{optimal path length}} \in [0, 1] \ . \tag{2}$$

We discretize the range of normalized solution qualities q into $Q + 1$ values.

The purpose of the normalization is to make performance histories across different problems comparable and thus enhance our ability to learn a stopping policy that generalizes across problems. Note that the worst path length is known and corresponds to the length of the first feasible path. The optimal path length, however, is unknown in a previously unseen problem and therefore must be predicted. We describe a CNN-based predictor for the optimal path length in Section V.

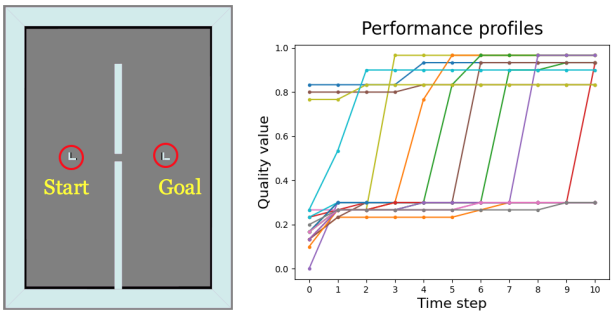
The history of normalized solution qualities H_i is the *performance profile*, represented by a vector of real values: $(q_0, \ldots, q_i)$. Note that the solution quality in a performance profile cannot decrease as the deliberation time increases and so $q_i \leq q_{i+1}$.

The utility function U encodes the user's trade-off between solution quality and computation time. The utility function $U(q, t)$ takes as input a normalized solution quality q and normalized time t and outputs a real-valued utility value. Although other functional forms for U are possible, we focus on a linear functional form, *i.e.*, $U(q, t) = wq - (1 - w)t$ where $w \in [0, 1]$ is a user-defined parameter. Thus, $U(q, t) \in [-1, 1]$ due to normalization. This linear functional form has been widely used in the meta-reasoning literature [8], [11]. Note that, although q always increases, utility may become negative as the computation time becomes large; this is the main motivation for meta-reasoning.

III. Motivating Example

Figure 2 illustrates a motivating example, in which we are using the anytime motion-planning algorithm RRT* [14] in a simple workspace that could occur as part of a larger domain. Due to the narrow passage located in the middle of the wall (*i.e.*, well-known challenge in sampling-based motion planning [15]), the algorithm tends to first find a longer path going through the wide passage and then later find a shorter path that passes through the narrow passage. Therefore, we expect that typical performance profiles would have a single jump when switching from the wide-passage path to the narrow-passage path (*i.e.*, when the homotopy class changes). However, other performance profiles are also possible because of the stochastic nature of the algorithm. For example, some paths may find the narrow passage first and some may not find the narrow passage within the allotted time. Even the length of the paths that pass through the same passage may also vary and the jump between the same two passages may occur at different times.

A sampling of 20 executions of the algorithm with the same start and goal configurations illustrates that the performance profiles are highly varying even in this simple example. Our goal is to design a predictable meta-reasoning framework that learns from such a distribution of performance profiles to find a utility-maximizing stopping policy.



(a) Example problem. (b) Plot of performance profiles.

Fig. 2. Motivating example where there are two passages in the 2D workspace: the narrow passage in the middle and the wide passage on the top that makes the robot detour around the wall. A rigid L-shaped robot is used here that can translate in the XY plane and rotate around the Z axis. Twenty performance profiles are obtained from twenty randomly generated paths by running the RRT* algorithm [14].

IV. Meta-Reasoning Frameworks

In both the model-based and model-free meta-reasoning frameworks, we assume we are given a dataset of representative problems. For model-based meta-reasoning, during training, we learn an MDP transition model over a state space that represents aspects of the performance profile so far. Then, we can compute a stopping policy via value iteration. This model-based learning and decision process is described in Sections IV-B. For model-free meta-reasoning, we directly learn a stopping policy, either in the form of a fully-connected neural network classifier operating on features of the performance profile or in the form of an RNN on the full performance profile; this process is described in Section IV-C. As we mentioned earlier, we also need a predictor during training and testing to estimate an optimal solution quality to be used as a normalizer; the predictor is described in Section V. Figure 3 shows the overall framework used for meta-reasoning. We first show how to generate a dataset of performance profiles before we present model-based and model-free meta-reasoning methods.

A. Dataset Generation

For each of the training problems, we run the given anytime motion planner for long enough to obtain (with high probability) the optimal quality solution. We use this optimal solution to train the optimal solution predictor (Section V). We also run the planner multiple times in each problem to gather a set of performance profiles, each of which consists of a temporal sequence of solution qualities $(q_0, \ldots, q_T)$. A set of performance profiles forms a training dataset of the model-based approach. For model-free meta-reasoning, however, we additionally need oracular decisions.

For each performance profile and a predetermined utility trade-off w that determines a functional form of the utility function, $wq_i - (1 - w)t_i$, we can compute a temporal sequence of optimal actions represented by $(a_0^*, \ldots, a_T^*)$,

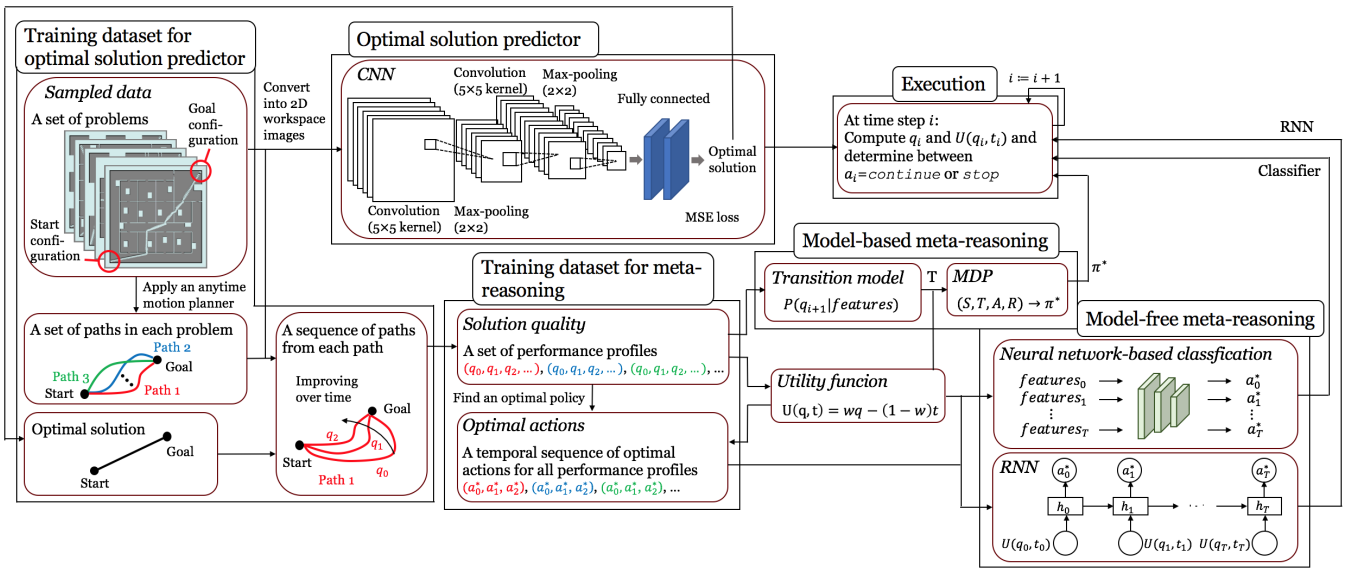


Fig. 3. Overall meta-reasoning framework.

where $a_i^* = \texttt{stop}$ for $i = \arg\max_{j \in [0,T]} U(q_j, t_j)$. Optimal actions before the time step i, $(a_0^*, \ldots, a_{i-1}^*)$, are `continue`, and those after the time step i, $(a_{i+1}^*, \ldots, a_T^*)$, are `stop`. The training dataset of the model-free approach consists of these paired performance profiles and optimal action sequences, and the parameter w, with multiple examples constructed from each training problem.

B. Model-Based Meta-Reasoning

We can formulate the model-based meta-reasoning problem as an MDP, $(\mathcal{S}, \mathcal{A}, T, R)$, with: state space $\mathcal{S}$; action space $\mathcal{A}$; transition model $T(s_i, a_i, s_{i+1}) = P(S_{i+1} = s_{i+1} | S_i = s_i, A_i = a_i)$ where $s_i, s_{i+1} \in \mathcal{S}, a_i \in \mathcal{A}$, and S_i, A_i are random variables denoting the state and action taken at time step i; and reward function $R(s_i, a_i) = r_i \in \mathbb{R}$.

In our setting, the state at time step i is typically defined to be a tuple of a normalized solution quality and normalized time, (q_i, t_i). We will also explore including additional features of the history in the state. The set of actions is $\{\texttt{continue}, \texttt{stop}\}$. The reward function provides a terminal reward based on the utility when the reasoning is stopped, so $R((q_i, t_i), a_i) = wq_i - (1 - w)t_i$ if $a_i = \texttt{stop}$ and 0 otherwise.

We first learn the transition model from the dataset of performance profiles. We then compute an optimal policy using this learned transition model.

Transition model When the state consists of (q, t) pairs, then, because time increments deterministically and sequentially, we only need to explicitly build a transition model of the form $P(q_{i+1} \mid q_i, t_i)$, where we arbitrarily set q_0 to be 0.

We note that, due to monotonicity in solution quality, $q_{i+1} \geq q_i$, and so we can model the distribution on the next quality value given that the solution quality at time step i is q_i, $P(q_{i+1} \mid q_i, t_i)$, as having a categorical or "multinoulli" distribution with parameter vector $\theta_{t,q}$ in the $(Q + 1 - Qq_i)$-simplex: that is, with $Q + 1 - Qq_i$ positive real parameters that sum to 1, so that $P(q_{i+1} \mid q_i, t_i) = \theta_{t,q}(Qq_{i+1} - Qq_i)$. Maximum likelihood estimates of the $\theta_{t,q}$ parameters can be computed by counting the number of the corresponding events in the data, and made more robust to sparse data by applying Laplace smoothing.

In general, the (q, t) values are not a sufficient representation of the latent state of the underlying planning process, and prediction accuracy for the next quality value can be improved by conditioning on additional features of the performance profile so far or, indeed, on the whole history $(q_0, \ldots, q_{i-1})$. However, increasing the number of conditioning variables substantially increases the number of parameters in the model and therefore the amount of data required to obtain accurate parameter estimates increases drastically.

Ultimately, the effectiveness of an estimated transition model is borne out in how well the decisions it induces perform at execution time. However, it is useful to be able to measure its accuracy as an independent component of the overall system. We use the *negative log likelihood* assigned by our estimated distribution $\widehat{\theta}$ to a set of held-out test data:

$$\mathcal{L}(\widehat{\theta}) = \sum_{d \in \mathcal{D}_{\text{test}}} -\log(\widehat{\theta}_d) \quad . \tag{3}$$

Model-based decision-making A policy π is a function that maps from state s to action a. We compute the policy that is optimal for the estimated transition model, π^*, by maximizing the expected discounted sum of the utility values over our decision horizon: $\pi^* = \arg\max_\pi \mathbb{E}\left[\sum_{i=0}^{T} R(s_i, a_i)\right]$. We solve this MDP using value iteration.

C. Model-Free Meta-Reasoning

In contrast with model-based meta-reasoning, learning a transition model is not required in the model-free approach. Moreover, the Markovian property assumed in an MDP may not hold for motion planning with non-smooth performance

profiles. Thus, we investigate the applicability of the model-free approach that learns a policy directly. In general, learning a policy requires reinforcement-learning methods, but we are able to take advantage of a special property of this meta-reasoning problem to reduce the policy learning to supervised learning.

We observe that if, on a training example problem, we run the anytime algorithm and compute the utility of stopping at each time step, we can find the optimal stopping time, t^* and therefore determine, for each time step i, that the optimal action a_i^* is `continue` if $t_i < t^*$ and `stop` otherwise. These labels constitute a "demonstration" of the optimal policy and so allow us to use them as targets for supervised learning. We explore two supervised learning methods: (1) a feed-forward neural-network classifier and (2) an RNN sequence-to-sequence model. The important distinction is that the RNN model does not require manual feature design.

Neural network-based classification From a training dataset consisting of paired performance profiles and optimal action sequences for a predetermined w, we design features that can be informative for predicting the optimal decision. Similar to model-based meta-reasoning, besides q and t, additional features of the history such as derivatives can be computed. We then learn a classifier to predict the optimal action at time t_{i+1} given features of the performance profile up through time t_i based on a set of labeled examples.

RNN sequence-to-sequence model Since our decision whether or not to stop deliberation depends on a sequence of previous observations, it is natural to frame the problem as one of mapping the performance profile so far, as a sequence of utility values to the sequence of actions. We can use an RNN model to learn this sequence-to-sequence mapping, which relieves us from having to hand-craft temporal features, as we did for the feed-forward network. The training data is a set of input-output pairs, where each pair consists of a temporal sequence of utility values, $U(q_i, t_i)\ \forall i \in [0, T]$, which can be obtained from performance profiles, as input and the corresponding temporal sequence of optimal actions, $a_i^*\ \forall i \in [0, T]$, as output for all performance profiles.

Because the input and output are low-dimensional, we use a simple RNN with ReLU nonlinearities and cross-entropy loss; we tune the number of hidden dimensions and learning rate using cross-validation. Although it might be better to use an LSTM, we get very good performance with this very simple model.

V. Optimal Solution Predictor

One difficulty with this meta-reasoning framework is that performance profiles from problems of different underlying difficulties are pooled together in the training data. In order to learn anything from that data and apply it to new problems, we would like to normalize any measured solution length in a problem by the *actual* shortest path length (length of the optimal solution) for that problem. Unfortunately, of course, we do not know that optimal path length, so we take the approach of learning to predict it from a description of the problem which, in our case, is a top-down 2D map of the environment.

From a known distribution of problems, we can sample a set of 2D workspace images. For each workspace we run the anytime motion planner for a long term and consider the resulting path length to be the optimum. The set of paired workspace images and optimal path lengths forms a dataset for a supervised regression problem. We train a CNN model to learn this mapping. We use squared loss and select the learning rate, the number of hidden units and batch size through cross-validation.

VI. Simulation Results

We perform three evaluation studies, measuring: (1) the effect of the CNN-based solution quality predictor; (2) the accuracy and computational requirements of transition-model estimation; and (3) the value of our approaches for meta-reasoning for anytime motion planning compared to several baselines. In all simulations, the robot (Figure 1) had three degrees of freedom in motion, that is, 2D translation and rotation. We defined the normalized quality q to be 30 discrete values (*i.e.*, $Q = 30$) and normalized running time t to be 200 discrete values (*i.e.*, $T = 200$). We set the utility trade-off w to be 0.8 unless stated otherwise. All simulations were implemented based on OMPL [16].

A. Solution Quality Predictor

We analyzed the performance of the CNN-based predictor both in terms of accuracy in predicting optimal path length and in terms of its effect on the overall quality of the meta-reasoning system. We trained a CNN model composed of two convolutional layers with (5×5) kernels interleaved with two (2×2) max-pooling layers, followed by fully-connected layers at the end.

1) Prediction Accuracy: We defined the prediction accuracy to be the percent similarity of the predicted path length to the optimal path length. To verify the accuracy of the predictor, we constructed an environment distribution by allowing the partial 2D workspace to randomly vary while the start and goal configurations are fixed. We then generated three different distributions having entirely different 2D workspaces. From each environment distribution, we sampled 1000 images (800 images are training data and 200 images are test data) whose resolution is 500×500. We ran RRT* on 800 images for long enough to obtain the shortest path lengths and used them as labels. The accuracy results in Table I were computed after applying a learned CNN model to 200 images that were held out. Figure 4 presents the longest and shortest paths from 200 images in each distribution. The results show that larger difference between the longest and shortest path lengths, the lower prediction accuracy. Nevertheless, the CNN-based predictor performed reliably with the amount of variation shown in Figure 4.

Environment type	Env. I	Env. II	Env. III
Accuracy	88.21%	96.23%	94.25%

TABLE I

Image resolution of environment images and accuracy results of the optimal solution predictor.

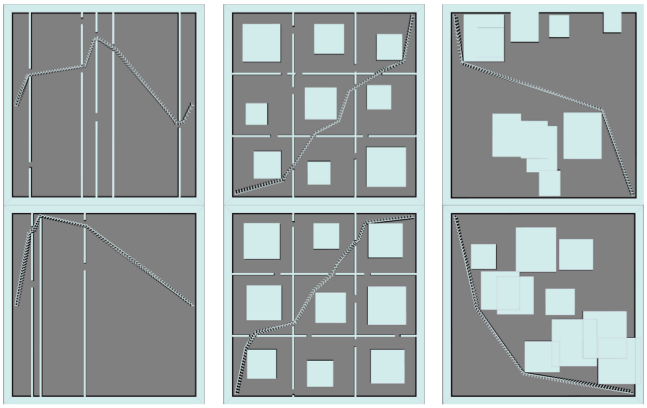


Fig. 4. The shortest (first row) and longest (second row) paths in three environment distributions (three columns). The lengths of the shortest and longest paths in meters are: 440.49 and 599.32 in the first column distribution, 641.37 and 757.61 in the second column distribution, and 622.91 and 832.13 in the third column distribution.

2) Effect of Prediction:

We validated the importance of having a reasonable predictor in meta-reasoning. To do that, we analyzed the consequence of using different normalizers when finding a stoppinging policy over the same environments above. We considered model-based meta-reasoning and RRT*. We learned a model from 800 workspaces and applied the model to compute the utility values on 200 held-out workspaces. We compared the CNN-based predictor with two other methods: a ground-truth method where the optimal solution was known; and a straight line connecting the start configuration to the goal configuration in 2D workspace which represents a coarse normalizer.

Table II presents the average predicted optimal path length and utility values computed over 200 workspaces for three normalizers. The results imply that poor prediction would decrease the performance of meta-reasoning drastically. The straight-line normalizer produced a comparable performance in the third environment because its prediction was close to the ground truth.

B. Transition Model Estimation

We evaluated the accuracy and efficiency of estimating the transition model, using 4500 trajectories as training data and 500 trajectories as test data obtained from five environment distributions (500×500 square meters) in Figure 1.

As using all possible features summarizing the entire history is infeasible, we introduce conditioning on two more features besides q and t providing a summary of the performance history: *slope* and *flatness*. The slope is just a one-step empirical derivative $(q_i - q_{i-1})/(t_i - t_{i-1})$ and flatness is the number of previous time steps for which the quality value has been equal to q_i.

Table III reports the results of estimating the quality transition model conditioned on different feature sets. The first row reports the data-likelihood loss (3) on the held-out data. Adding the slope and flatness features substantially improved the model's predictions. Unfortunately, as indicated in the second row, which reports the time required to estimate the model parameters, the improvements in accuracy come at a huge computational cost. This is the curse of dimensionality arising from estimating a large discrete model.

C. The Value of Model-Based Meta-Reasoning for Anytime Motion Planning

We applied the proposed model-based and model-free meta-reasoning frameworks to three anytime motion planners: RRT*, PRM* [14], [17] and Lazy PRM* [18]. We evaluated their effectiveness in five different environment distributions from which we can sample 2D workspaces whose area is 500×500 square meters (Figure 1). We generated a dataset from these distributions and extracted 2D workspace images to learn a CNN-based predictor. All training was based on datasets consisting of 4500 training performance profiles and 500 test performance profiles. We ran the planners 20 times on every held-out workspace to measure the performance of the meta-reasoning methods. In RRT*, we set the range parameter to be large (*i.e.*, 100) so that the planner found a first feasible path quickly but its initial quality was poor. Examples of the profile of the utility values are shown in Figure 5.

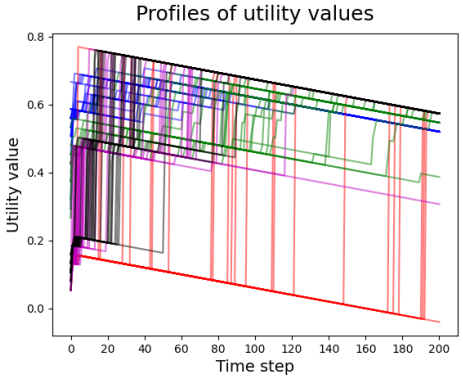


Fig. 5. Profiles of utility values obtained by running RRT* over five different environment distributions. Different colors represent different environment distributions.

We compared the performance of the proposed methods with other baselines in terms of the mean and 95% confidence interval of utility values from held-out workspaces while varying the value of w. The oracle outputs the optimal actions from all test performance profiles, giving an upper bound on the mean utility value.

We additionally designed two simple meta-reasoning approaches as baselines: the *fixed-time strategy* and the *fixed-quality strategy*. The fixed-time strategy finds a single time step $i \in [0, T]$ by averaging the time steps at which the optimal `stop` action is taken from all training performance profiles, where each optimal time step can be computed by: $\arg\max_{j \in [0,T]} U(q_j, t_j)$. Then, in test performance profiles, the fixed-time strategy stops at the chosen time step i. Likewise, the fixed-quality strategy finds an unnormalized quality value $Qq \in [0, Q]$ by averaging the unnormalized quality values at which the optimal `stop` action is taken from all training performance profiles. The fixed-quality

Environment type	Average path length			Utility		
	Ground truth	CNN	Straight line	Ground truth	CNN	Straight line
Environment I	592.42	**542.86**	440.00	0.63±0.09	**0.58±0.08**	0.39±0.08
Environment II	701.67	**716.72**	622.25	0.70±0.00	**0.65±0.11**	0.40±0.00
Environment III	694.68	**683.66**	622.25	0.67±0.00	**0.66±0.01**	0.60±0.09

TABLE II

AVERAGE OPTIMAL PATH LENGTH PREDICTED BY THREE NORMALIZERS AND UTILITY VALUES OBTAINED BY APPLYING THE NORMALIZERS TO META-REASONING. UTILITY VALUES ARE REPRESENTED BY MEAN AND 95% CONFIDENCE INTERVAL. THREE ENVIRONMENTS ARE THE SAME AS USED IN TABLE I.

Features	No features	t_i	t_i, q_i	t_i, q_i, q_{i-1}	t_i, q_i, slope	$t_i, q_i, \text{slope}, \text{flatness}$	$t_i, \text{slope}, \text{flatness}$
Loss	1.84	1.79	0.22	0.19	0.10	**0.03**	1.79
Time (s)	0.41	2.53	10.64	2858.63	11933.72	–	–

TABLE III

DATA-LIKELIHOOD LOSS, ACCURACY, AND COMPUTATION TIME OF ESTIMATED TRANSITION MODEL FOR DIFFERENT FEATURE SETS. THE SYMBOL "–" IMPLIES THE TIME TAKEN MORE THAN A DAY.

strategy stops when the performance profile reaches the unnormalized value of Qq in all test performance profiles.

Although we demonstrated that adding features to the model-based method can substantially improve its accuracy, the computational requirements of doing so were impossible for our larger experimental domains. For this reason, we used (q_i, q_{i-1}, t) as features for the model-based method. However, $(q_i, q_{i-1}, t, \text{slope}, \text{flatness})$ were used for classification. For the classification, we trained a neural network composed of three layers (128, 64, 32 neurons per each layer) and the ReLU activation functions. For the RNN model, we used 300 neurons in the hidden layer, 0.0001 as a learning rate, and 40000 epochs.

The results are shown in table IV. In general, the model-based meta-reasoning methods provide a substantial advantage over the baselines. The most reliably good results come from the RNN-based predictor, which is able to discover its own temporal features for prediction and does not suffer from estimation problems due to model size, which cause difficulty for the model-based methods, particularly when increased emphasis is placed on time cost by decreasing w.

VII. RELATED WORK

The study of meta-reasoning has its roots in the late 80's in the problem of building intelligent decision-making agents under bounded rationality [8] in terms of two equivalent models: anytime algorithms [6] and flexible computation [7]. Here we will focus on meta-reasoning in the context of planning. For more complete surveys, see [9], [19], [20], and [21].

Meta-reasoning has been extensively employed for online planning to decide when to switch to execution during concurrent planning and execution. In online planning, the agent must plan ahead for a bounded time before taking action. The longer the agent spends planning, the higher performance it can achieve but the higher cost it pays, such as wasting energy. Hay *et al*. [22] applied meta-reasoning as a part of the selection process in Monte-Carlo tree search and compared that with the bandit setting. Lin *et al*. [23] proved the hardness of solving the meta-reasoning version of MDPs and instead proposed an approximation algorithm. Other planning domains have also adopted meta-reasoning, including algorithm selection in sorting [24] and belief space meta-reasoning in autonomous driving applications [25].

In this work, our interest lies in anytime motion planning in continuous configuration spaces, but there exist previous works on meta-reasoning for path planning in discrete domains. O'Ceallaigh and Ruml [26] designed a meta-reasoning-based search algorithm that dynamically switches between greedy hill-climbing for committing to actions and A^* for deliberation. Cserna *et al*. [27] considered durative actions and showed that taking a slower action can lead to a better outcome by allowing a longer time for planning. In their subsequent work [28], they presented several backup methods to estimate heuristic values in order to decide which node to expand in tree search.

The most related work to ours is by Hansen and Zilberstein [11], where they proposed a dynamic-programming approach to solve the model-based variant of meta-reasoning. They later developed online approaches, such as online performance prediction [10] and RL-based model-free meta-reasoning [12] in an attempt to remove the necessity of preprocessing and gathering data. They particularly considered the solution quality to be safety, yielding smooth performance profiles. However, we deal with non-smooth performance profiles due to our definition of path length-based solution quality in this work. As a consequence, a learning framework is necessary for anytime motion planning.

While algorithmic properties like feasibility and completeness have drawn much attention in anytime motion planning work [5], bounded rationality has generally been overlooked. However, recent work by Sudhakar *et al*. [29] formulated an energy-minimization problem that trades off actuation energy and computing energy in anytime motion planners for low-power robotic platforms. Their approach exhibits meta-reasoning in terms of energy consumption although learning from data to help decision-making was not considered.

VIII. CONCLUSION

In this work, we introduce a general-purpose meta-reasoning framework for anytime motion planning. Sev-

Anytime planner	Utility trade-off (*i.e.*, w)	Meta-reasoning method											
		Oracle		Model-based		Classification		RNN		Fixed time		Fixed quality	
		Mean	CI	Mean	CI	Mean	CI	Mean	CI	Mean	CI	Mean	CI
RRT*	0.8	0.677	0.008	**0.663**	**0.010**	**0.658**	**0.014**	**0.670***	**0.006***	0.575	0.016	0.634	0.011
	0.6	0.442	0.008	**0.414***	**0.012***	**0.399**	**0.013**	**0.401**	**0.013**	0.341	0.014	0.337	0.016
	0.4	0.244	0.006	0.168	0.015	**0.195***	**0.008***	**0.191**	**0.010**	0.176	0.008	0.010	0.019
	0.2	0.096	0.004	0.031	0.016	**0.079***	**0.004***	**0.074**	**0.004**	**0.072**	**0.004**	-0.283	0.031
PRM*	0.8	0.718	0.002	**0.702***	**0.003***	**0.697**	**0.003**	**0.698**	**0.003**	0.662	0.004	0.674	0.004
	0.6	0.507	0.002	**0.490***	**0.004***	**0.489**	**0.003**	**0.485**	**0.004**	0.451	0.004	0.396	0.008
	0.4	0.314	0.002	**0.297***	**0.004***	**0.292**	**0.003**	**0.294**	**0.003**	0.271	0.003	0.118	0.011
	0.2	0.140	0.001	**0.120**	**0.004**	**0.125**	**0.002**	**0.125***	**0.002***	0.120	0.002	-0.344	0.016
Lazy PRM*	0.8	0.643	0.006	**0.591**	**0.009**	**0.593***	**0.008***	**0.589**	**0.008**	0.559	0.007	**0.588**	**0.009**
	0.6	0.441	0.005	**0.376**	**0.010**	**0.387**	**0.006**	**0.393***	**0.007***	0.361	0.006	0.302	0.010
	0.4	0.266	0.003	0.201	0.009	**0.224***	**0.004***	**0.221**	**0.005**	0.216	0.003	0.0157	0.012
	0.2	0.117	0.002	0.053	0.009	**0.100**	**0.002**	**0.100***	**0.001***	**0.100**	**0.002**	-0.367	0.016

TABLE IV

COMPARISON WITH BASELINES FOR THREE DIFFERENT ANYTIME MOTION PLANNERS. IN EACH ROW, WE MARK WITH $*$ THE METHOD WHOSE PERFORMANCE IS CLOSEST TO THE ORACLE. CI IMPLIES THE 95% CONFIDENCE INTERVAL. BOLDED NUMBERS REPRESENT THE METHODS WHOSE PERFORMANCE IS NOT STATISTICALLY SIGNIFICANTLY DIFFERENT FROM THE METHOD WITH $*$.

eral data-driven learning methods are proposed, namely model-based and model-free meta-reasoning. We show that learning-based meta-reasoning approaches can improve the utility of anytime motion-planning in embedded settings where computation time must be traded off against execution time.

REFERENCES

[1] S. Zilberstein, "Using anytime algorithms in intelligent systems," *AI magazine*, vol. 17, no. 3, pp. 73–73, 1996.

[2] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1478–1483.

[3] R. Shome and K. E. Bekris, "Anytime multi-arm task and motion planning for pick-and-place of individual objects via handoffs," in *2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*. IEEE, 2019, pp. 37–43.

[4] N. Shah, D. K. Vasudevan, K. Kumar, P. Kamojjhala, and S. Srivastava, "Anytime integrated task and motion policies for stochastic environments," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9285–9291.

[5] J. D. Gammell and M. P. Strub, "Asymptotically optimal sampling-based motion planning methods," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4.

[6] T. L. Dean and M. S. Boddy, "An analysis of time-dependent planning." in *AAAI*, vol. 88, 1988, pp. 49–54.

[7] E. J. Horvitz, "Computation and action under bounded resources," Ph.D. dissertation, Citeseer, 1990.

[8] S. J. Russell and E. Wefald, *Do the right thing: studies in limited rationality*. MIT press, 1991.

[9] R. Ackerman and V. A. Thompson, "Meta-reasoning: Monitoring and control of thinking and reasoning," *Trends in Cognitive Sciences*, vol. 21, no. 8, pp. 607–617, 2017.

[10] J. Svegliato, K. H. Wray, and S. Zilberstein, "Meta-level control of anytime algorithms with online performance prediction," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 2018.

[11] 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.

[12] J. Svegliato, P. Sharma, and S. Zilberstein, "A model-free approach to meta-level control of anytime algorithms," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 11 436–11 442.

[13] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.

[14] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[15] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, S. Sorkin *et al.*, "On finding narrow passages with probabilistic roadmap planners," in *Robotics: the algorithmic perspective: 1998 workshop on the algorithmic foundations of robotics*, 1998, pp. 141–154.

[16] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[17] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[18] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 2951–2957.

[19] M. T. Cox, "Metacognition in computation: A selected research review," *Artificial intelligence*, vol. 169, no. 2, pp. 104–141, 2005.

[20] M. L. Anderson and T. Oates, "A review of recent research in metareasoning and metalearning," *AI Magazine*, vol. 28, no. 1, pp. 12–12, 2007.

[21] T. L. Griffiths, F. Callaway, M. B. Chang, E. Grant, P. M. Krueger, and F. Lieder, "Doing more with less: meta-reasoning and meta-learning in humans and machines," *Current Opinion in Behavioral Sciences*, vol. 29, pp. 24–30, 2019.

[22] N. Hay, S. Russell, D. Tolpin, and S. E. Shimony, "Selecting computations: theory and applications," in *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, 2012, pp. 346–355.

[23] C. H. Lin, A. Kolobov, E. Kamar, and E. Horvitz, "Metareasoning for planning under uncertainty," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[24] F. Lieder, D. Plunkett, J. B. Hamrick, S. J. Russell, N. Hay, and T. Griffiths, "Algorithm selection by rational metareasoning as a model of human strategy selection," in *Advances in neural information processing systems*, 2014, pp. 2870–2878.

[25] J. Svegliato, K. H. Wray, S. J. Witwicki, J. Biswas, and S. Zilberstein, "Belief space metareasoning for exception recovery," in *IROS*, 2019.

[26] D. O'Ceallaigh and W. Ruml, "Metareasoning in real-time heuristic search," in *Eighth Annual Symposium on Combinatorial Search*, 2015.

[27] B. Cserna, W. Ruml, and J. Frank, "Planning time to think: Metareasoning for on-line planning with durative actions," in *Twenty-Seventh International Conference on Automated Planning and Scheduling*, 2017.

[28] A. Mitchell, W. Ruml, F. Spaniol, J. Hoffmann, and M. Petrik, "Real-time planning as decision-making under uncertainty," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 2338–2345.

[29] S. Sudhakar, S. Karaman, and V. Sze, "Balancing actuation and computing energy in motion planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4259–4265.