2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

Anticipatory Planning: Improving Long-Lived Planning by Estimating Expected Cost of Future Tasks

Roshan Dhakal, Md Ridwan Hossain Talukder and Gregory J. Stein

***Abstract*— We consider a service robot in a household environment given a sequence of high-level tasks one at a time. Most existing task planners, lacking knowledge of what they may be asked to do next, solve each task in isolation and so may unwittingly introduce side effects that make subsequent tasks more costly. In order to reduce the overall cost of completing all tasks, we consider that the robot must anticipate the impact its actions could have on future tasks. Thus, we propose *anticipatory planning*: an approach in which estimates of the expected future cost, from a graph neural network, augment model-based task planning. Our approach guides the robot towards behaviors that encourage preparation and organization, reducing overall costs in long-lived planning scenarios. We evaluate our method on blockworld environments and show that our approach reduces the overall planning costs by 5% as compared to planning without anticipatory planning. Additionally, if given an opportunity to *prepare* the environment in advance (a special case of anticipatory planning), our planner improves overall cost by 11%.**

I. Introduction

We consider a service robot in a household that is given a sequence of high-level task specifications (task planning problem) one at a time, each expressed in Planning Domain Definition Language (PDDL) [1] by a human operator. Most existing task planners [2]–[9] in this setting solve each task objective one at a time in isolation to complete each in minimum cost. However, since the environment persists between tasks, the robot may unwittingly introduce problematic side effects for the subsequent tasks it has yet to be assigned. For example (Fig. 1), consider a household robot given a task from a sequence of tasks to clear the dining table that has a plate on it. Though the quickest solution is to move the plate to the desk, this limits the use of the desk later and increases the planning cost of a future task to clean the bowl, and so should instead be placed in the kitchen sink.

If the robot is to minimize an overall cost over *the entire sequence of tasks*, it must have the capacity to understand the impact of its actions—those that solve the current task—on future tasks. In other words, to reduce the overall cost, it would require the robot to act in a way that seeks to pay down both the immediate cost of completing a current task and the expected cost associated with future tasks in the sequence which we call the *anticipatory cost*. We introduce *anticipatory planning*: planning so as to minimize the joint objective of planning cost for the current task and the anticipatory cost.

Roshan Dhakal, Md Ridwan Hossain Talukder and Gregory J. Stein are affiliated with the CS Department, George Mason University, Fairfax, VA: {rdhakal2, mtalukd, gjstein}@gmu.edu

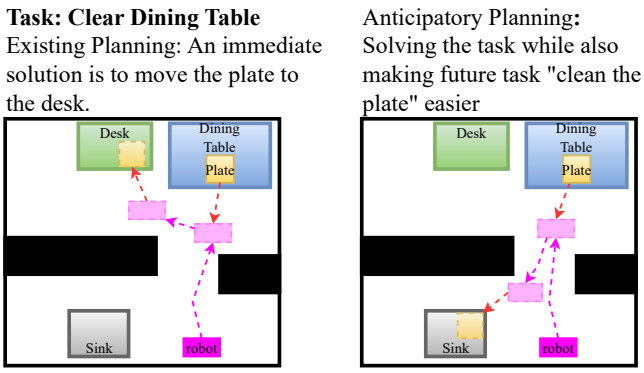


Fig. 1: **An Anticipatory Planning Scenario**: The immediate plan to clear the dining table (left) is to move away the plate to the desk with a minimum cost. However, this limits the use of the desk later and also increases the cost of cleaning the bowl later. Our robot instead anticipates how it may later interact with the environment and takes preemptive action (right) to reduce expected future cost.

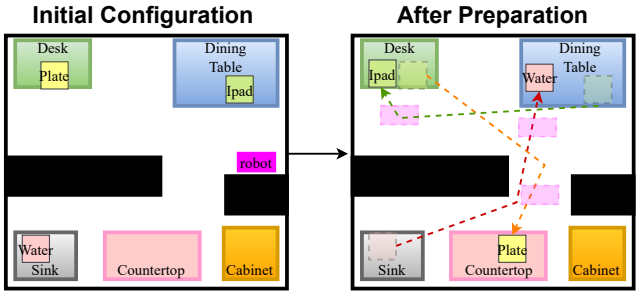
If the robot were provided all its tasks in advance, existing task planners could be used to find the minimum-total-cost plan. Though the robot will not typically know future tasks in advance, we assert that there exists a *task distribution* inherent to the environment that can be used to define the *anticipatory cost*: the *expected* future cost. To plan well, the robot must determine which actions will reduce the joint cost of completing the current task and the anticipatory cost. Given access to task distribution and sufficient computation, the robot could reduce anticipatory cost. However, during deployment, computing anticipatory cost is often computationally infeasible, as there could be hundreds of possible future tasks, or because the robot lacks direct access to the task distribution when deployed in its new environment.

So as to overcome the challenges associated with limited knowledge or computation, we use learning to estimate the anticipatory cost. Doing so allows us to transfer experience from an offline training phase—when we have both sufficient computation and direct access to the task distribution—to improve planning performance during deployment. Offline, the robot has direct access to the task distribution, from which it generates data to train a Graph Neural Network [10]–[12] to estimate the anticipatory cost. During deployment, the robot uses this *anticipatory cost estimator* to augment model-based planning and improve overall cost.

Even when a task is not given, our Anticipatory Planning approach allows our robot to ready the environment in anticipation of future tasks, reducing cost for when a task

979-8-3503-2365-8/23/$31.00 ©2023 IEEE

Our robot prepares an environment for probable future tasks by minimizing the expected cost associated with them.



Probable Future Tasks:
Clear the dining table, clean the plate, put the plate in the cabinet, bring water to the desk, move water to the dining table.

Fig. 2: **Preparation as Task-Free Anticipatory Planning**: The left figure shows the current state of an environment with probable future tasks. In the right figure, our robot anticipates future tasks and prepares an environment to minimize the overall planning cost of a future task.

is eventually provided. We refer to this task-free planning setting as *preparation*, illustrated in Fig. 2.

To evaluate Anticipatory Planning, averaged across an ensemble of multi-task sequences in a blockworld environment. We demonstrate that our approach improves overall cost by 5%. We also demonstrate that even in the absence of an immediate task, anticipatory planning allows the robot to *prepare* the environment and reduce the overall costs of potential future tasks by 11%.

II. Related Work

Task Planning and Learning-Augmented Planning: Task and Motion Planning (TAMP) involves jointly solving a high-level task and the motion planning to accomplish those tasks [13].[1] Many existing works in this space [2]–[6], [9] are focused on solving a TAMP problem as quickly as possible by using so-called classical planning strategies.

Recently, learning has been used to accelerate planning, extending the limits of plan complexity and improving plan quality. Learning techniques have been integrated into many aspects of TAMP systems, from learning samplers for continuous values [7], [8], [14], [15] to learning guidance for symbolic planning [16]. However, approaches for both *classical TAMP* and *learning for TAMP* primarily focus on solving small-scale or deterministic planning problems and are not directly applicable to solving our anticipatory planning objective.

Learning from Demonstration and Common Sense Planning: Common sense reasoning describes a class of scenarios in which robot behavior is expected to agree with human intuition, though so far has generally defied precise mathematical definition. Agrawal [17] identifies under-specification in the task definition as limiting common sense and points to approaches to transfer and meta-learning [18], [19] or diverse skill learning [20] that have shown limited promise in addressing this under-specification. In the context of service and assistive robots, approaches such as learning from demonstration (LfD), inverse reinforcement learning [21], reward learning [22], and similar [23] have proven effective in situations where an explicit reward function can be difficult to write down, encouraging intrinsic motivation [24], [25], courtesy [26], [27], and avoiding side effects [28], [29] in service-like domains. These approaches to improve robot behavior typically struggle for long-horizon planning in non-deterministic settings [30] and so efforts in this space are primarily limited to deterministic settings—for low-level [31]–[34] and high-level [35], [36] planning and from physical corrections [37], [38]—or short-time-horizons [39], [40].

Graph Neural Networks and Applications to TAMP: In this work, we use graph neural networks (GNNs) [10], [12], [41]. See [11] for a recent survey. Recently, GNNs have proven effective for geometric integrated TAMP problems. Graph representations of the state of an environment can be fed into a graph neural net to guide sample-based motion planning [16] or to estimate the relative importance of objects in a scene to accelerate search over tasks [42]. We rely on the powerful representational capacity and generalization capabilities of GNNs to estimate the anticipatory cost in an effort to improve planning across multiple tasks.

III. Problem Statement

A. Anticipatory Planning

We want to reduce the overall cost of solving a sequence of N tasks τ, given only one task at a time considering both the immediate cost of completing the current task and the anticipatory cost of a state for future tasks in the sequence. If we were to know the sequence of tasks in advance, we could find a plan that minimizes total cost according to

$$s^*_{g_1}, \cdots, s^*_{g_N} = \underset{s_{g_i} \in G_{\tau_i} \forall i \in 1 \cdots n}{\operatorname{argmin}} \left[V^*_{s_{g_1}}(s_0) + V^*_{s_{g_2}}(s_{g_1}) + \cdots + V^*_{s_{g_n}}(s_{g_{n-1}}) \right] \tag{1}$$

where τ_i is i^{th} task in sequence, $V^*_{s_{g_{i+1}}}(s_{g_i})$ is the minimum cost of completing τ_i, which can be computed via any deterministic planner, for example, Sec. IV. State s_{g_i} is the final state of completing τ_i and also the initial state for completing task τ_{i+1}. However, since the robot will in general not know all the tasks in advance, it instead seeks a policy that minimizes *expected* total cost, where future tasks τ are drawn from a *task distribution*: $\tau \sim P(\tau)$.

Reasoning about future cost over a long sequence of tasks is computationally challenging, and so we instead seek to minimize the expected cost associated with an immediately available task and a *single* next task in the sequence. In general, planning in this domain involves first moving from starting state s_0 to some intermediate goal state s', belonging

[1] We do not currently tackle *Integrated* Task and Motion Planning. As we highlight in Sec. IV, low-level motion planning and the costs of motion-borne actions are computed independently of high-level task planning.

to the set of goal states G_τ in which the task τ is completed, and then subsequently completing another task $\tau' \sim P(\tau)$. We refer to this planning objective as *anticipatory planning*, the aim of which is to find the intermediate goal state s_g^* that minimizes the total cost according to

$$\begin{aligned} s_g^* &= \underset{s'_g \in G_\tau}{\operatorname{argmin}} \left[V^*_{s'_g}(s_0) + \sum_{\tau'} P(\tau') V^*_{\tau'}(s'_g) \right] \\ &= \underset{s'_g \in G_\tau}{\operatorname{argmin}} \left[V^*_{s'_g}(s_0) + V^*_{A.P.}(s'_g) \right] \end{aligned} \tag{2}$$

where $V^*_{s'_g}(s_0)$ is the optimal cost of moving from state s_0 to state s'_g, and $V^*_{\tau'}(s'_g)$ is the cost of completing task τ' starting from state s'_g. We refer to $V^*_{A.P.}(s'_g)$ as the *anticipatory planning cost*: the expected cost of completing a single follow-up task starting from state s'_g.

$$V^*_{A.P.}(s) = \sum_{\tau'} P(\tau') V^*_{\tau'}(s) \tag{3}$$

While $V^*_{s'_g}(s_0)$ can be computed using existing task planning solvers, the anticipatory planning cost is often too difficult to compute online, and so we instead estimate it via learning (see Sec. V).

B. Preparation as Task-Free Anticipatory Planning

Even when the robot is not assigned an immediate task τ—a common scenario for household robots, which may not be constantly given active instruction—anticipatory planning can inform how the robot can reconfigure its environment so as to reduce expected future cost for when it is eventually given a task. We refer to this objective as *preparation*, in which the robot seeks to find a prepared state of the environment s^*_{prep}, when the state has the minimum expected cost $V^*_{A.P.}$. This scenario is related to the anticipatory planning objective of Eq. (2), yet the robot (not given an immediate task) accumulates no cost for immediate actions. As such, it is the robot's objective to minimize only the anticipatory planning cost $V^*_{A.P.}$:

$$s^*_{prep} = \underset{s \in S}{\operatorname{argmin}} \left[V^*_{A.P.}(s) \right] \tag{4}$$

where S is the set of all possible states in the environment.

In our experiments, we will demonstrate the effectiveness of both anticipatory planning and preparation in improving the expected cost of planning.

IV. Preliminaries: Task Planning with PDDL

In this paper, we consider pick-and-place-style task planning problems, for which action costs are computed separately via low-level motion planning. To achieve high-level goals—e.g., clearing a table—robots need to be able to carry out high-level actions, such as picking up a bowl and moving around the environment. We represent both the tasks and the actions available to the robot for this Task Planning problem using the Planning Domain Definition Language (PDDL) [1]. PDDL defines a fully observable deterministic

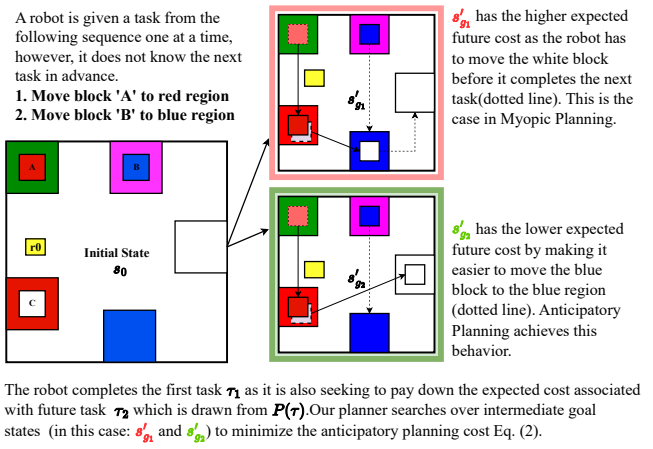


Fig. 3: A schematic overview: Our robot reduces the overall planning cost of the sequence by considering the immediate cost of the current task and the expected cost associated with future tasks.

Listing 1: PDDL code definition for action `pick`

```
(:action pick
  :parameters (?r ?b ?s)
  :precondition (and (Robot ?r) (HandEmpty
      ?r) (At ?r ?s) (In ?b ?s))
  :effect (and (Holding ?r ?b) (not (In ?b
      ?s))
               (not (HandEmpty ?r))
               (increase (total-cost) 100)
               ))
```

planning problem as tuple (A, s_0, g), where A is a set of parameterized actions, s_0 is an initial state and g is a goal condition for the problem. An example PDDL definition of a `pick` action is included in Listing 1.

Actions involving picking up or placing an object have a constant cost. By contrast, costs for the $move$ action come from a separate motion planning process. We consider a $move$ action as the robot's motion from one location $l1$ to another $l2$, represented by the tuple $(l1, l2, f)$ where f is a collision checking function between $l1$ and $l2$. We use the Lazy Probabilistic Roadmap (Lazy PRM) algorithm [43] to get the cost for $move$ action provided by [2]. Lazy PRM operates like traditional PRM [44], but lazily checks the collision, which removes redundant sampling of regions that does not provide the solution and accelerates planning.

We use Fast Downward [45] for task planning, using A* search with the admissible `max` heuristic, so as to yield optimal plans.[2] The planner consumes in the PDDL definition of the problem, including pre-computed motion planning costs for `move` actions, and returns minimum-cost plans that solve the task. We use the notation $V^*_{s_g}(s_0)$ to mean "the cost of the optimal plan to get from state s_0 to

[2]We note that one could use an inadmissible search heuristic—e.g., the well-known Fast-Forward (FF) heuristic [46]—for both data generation and planning and trade optimality for improvements in planning speed. In this work, we focus only on admissible search via A*.

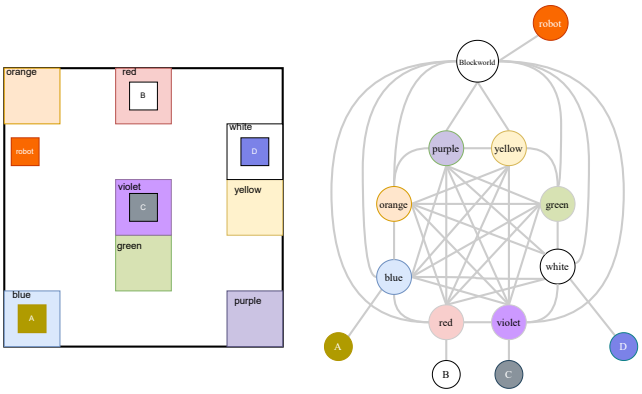


Fig. 4: An example of a state in an environment and its corresponding graph structure for training data

goal state s_g."

V. Estimating Anticipatory Cost

Direct computation of the anticipatory cost $V^*_{A.P.}(s)$ during deployment is often not feasible—either because it is computationally challenging or because the robot does not have direct access to the task distribution $P(\tau)$. This makes the robot unable to search over the states during planning to solve Eq. (2), which depends on the anticipatory cost $V^*_{A.P.}(s)$. To overcome this limitation, we instead use learning to estimate the anticipatory cost of the state s. Offline, we randomly generate task sequences drawn from the task distribution $P(\tau)$, with which we compute samples of the anticipatory cost to serve as training data.

We learn the anticipatory cost in a supervised manner. To be useful as a component in a search procedure, we require a process for mapping states to inputs to our estimator. During deployment, a photorealistic simulator may not be available, so images are not well-suited for this purpose. Instead, we represent the environment state as a graph and train a *Graph Neural Network* [11] to estimate the anticipatory cost. Graph Neural Networks (GNNs) have proven effective tools for making predictions in household robotics settings [16], [42].

A. Representing the Environment State as a Graph

The state s of an environment $\mathcal{E}$ defines the location of each object in that environment as well as the location of various regions in which those objects can be placed. Each graph $\mathcal{G}$ represents a state. For example, in the blockworld environment, as shown in Fig. 3, the state is defined as blocks being placed in specific regions.

To represent the environment state as a graph G, nodes correspond to (i) places in the environment (including a node for the environment itself), (ii) objects in the environment, and (iii) locations in which objects can be placed. Edges are created whenever one entity is contained by another. For example, a block is in the red region, so an edge connects the two. An illustration of an example state and its corresponding graph can be seen in Fig. 4.

To act as input to a graph neural network, nodes have accompanying *node input features* that include properties of the entity it represents. Node features in our graph consists of a one hot encoding of its entity type (whether it is a room, location, or object), entity color (specifying its semantic class), and the 2D coordinate of its location in the environment. For example, a node for a `book` from Fig. 4 has the entity type is `object`, thus we get one hot encoding of `object`. Since it is not both room and location type, all vectors in both encodings are zeros. `Book` is an object type from which we get the corresponding encoding. It also has a color `blue` which has scalar value $(0.0, 0.0, 1.0, 1.0)$.

Algorithm 1 Search for an Intermediate State

function SEARCH(s_0, ESTIMATOR, τ)
  $\pi, V^*_\tau(s'_g) = \text{TASKPLAN}(s_0, \tau)$
  $s^* = s'_g$
  $V^*_{\text{total}}(s^*) = V^*_{s'_g}(s_0) + \text{ESTIMATOR}(s'_g)$
  $G_\tau = \text{SETOFALTERNATEGOALSTATES}(\tau, \pi)$
  for $s' \in G_\tau$ **do**
    $V^*_{\text{total}}(s') = V^*_{s'}(s_0) + \text{ESTIMATOR}(s')$
    if $V^*_{\text{total}}(s') \leq V^*_{\text{total}}(s^*)$ **then**
      $V^*_{\text{total}}(s^*) = V^*_{\text{total}}(s')$
      $s^* = s'$
  return s^*

B. Data Generation

To train our graph neural network (GNN) to estimate the anticipatory cost, we require training data generated from each environment class of interest. To obtain this data, we use Sec. IV to solve each planning problem instance from task distribution $P(\tau)$ from randomly sampled 200 states in each seed of environment. We generate data from 250 procedurally-generated training environments and 150 distinct testing environments. We compute the anticipatory cost $V_{A.P.}$ for each state s. Each labeled datum consists of the graph $\mathcal{G}$ of s and its anticipatory cost $V_{A.P.}$.

C. Graph Neural Network Structure and Training

Our estimator is a graph neural network that takes as input a graph $\mathcal{G}$ with nodes v and edges $e(v1, v2)$ corresponding to the environment state s trained to estimate the anticipatory cost $V^*_{A.P.}(s)$. Our graph neural network consists of three SAGEConv layers [41], provided by the PyTorch Geometric package [47], each followed by leaky ReLU activation function. We also add a mean pooling layer for batch-wise graph level outputs averaging node features and, finally, a linear classifier layer to produce the anticipatory cost. We use mean absolute error (MAE) as our loss function. We train our model using Adam optimizer [48] on PyTorch [49] with a batch size of 8 and roughly using 50k training samples. We train for 10 epochs using a fixed learning rate of 0.01.

VI. Search In Anticipatory Planning

It is computationally infeasible to enumerate all states in a complex environment when searching during anticipatory

planning to find states that reduce overall cost according to Eq. (2). Instead, we rely on an iterative procedure to search more efficiently over states likely to improve the total cost, shown in Alg. 1.

We first compute the minimum-cost plan for task τ using the solver, TASKPLAN (Sec. IV), which returns both the plan π and the immediate cost of that plan $V^*_\tau(s'_g)$. We compute the total expected cost of the state, including the estimated expected future cost via ESTIMATOR(s'_g). From this plan, we compute a set of possible *alternate goal states* G_τ via a procedure SETOFALTERNATEGOALSTATES: each element $s_g \in G_\tau$ is a potential state in which the task τ is satisfied. To save computation, the procedure SETOFALTERNATEGOALSTATES is not exhaustive, and instead enumerates alternate placements only of objects manipulated during solution plan π, keeping only those that still solve the task τ. For example, in Fig. 3, for the first task, the immediate goal state is block `A` being in the red region and block `C` being in blue. One such alternate goal state could be to place block `A` in the red region and `C` in the white region. We iterate over all $s_g \in G_\tau$ to find the alternative intermediate goal state that minimizes the total cost—computed via both the Fast Downward solver (Sec. IV) and our ESTIMATOR for the anticipatory cost.

Similarly, for our *preparation* tasks, for which a current task is not given, planning via Eq. (4) involves minimizing the anticipatory cost over state space. This algorithm follows a hill-climbing optimization approach. First, the algorithm is seeded with the initial state s_0. For 200 iterations, the robot randomly samples tasks from the task distribution and evaluates whether the anticipatory cost $V_{A.P.}$ improves along the way to solving it, computed using the trained model from Sec. V. If the anticipatory cost of the resulting state is lower than that of the previously visited state, the robot chooses that state as the prepared state ($s^*_{A.P.}$) and repeats.

VII. EXPERIMENTS AND RESULTS

We evaluate our Anticipatory Planning approach on a 2D Blockworld environment, with procedurally generated worlds as shown in Fig. 5. We include experimental trials showing both (i) planning beginning from a random initial configuration of the environment and also (ii) the impact of *preparation*: task-free anticipatory planning (as described in Sec. III-B).

For this environment, the task distribution is hand coded and has 20–25 pick-and-place-style tasks for each environment. Task specifications consist of placement directives for one or two blocks; for example, a task could involve moving only block A to the red region or moving block A and block B to the red and blue region respectively. The tasks are randomly sampled in a way that they are achievable in the environment and that only non-white blocks and non-white regions are involved. We also note that multiple blocks cannot be placed in one location. We use a uniform distribution over possible tasks. We then generate training data as described in Sec V-B.

Approach	Average Cost
N.L. Myopic	827.8
A.P. (ours)	**785.1**

(a) Anticipatory Planning

Approach	Average Cost
Prep (ours) + N.L.Myopic	779.0
Prep (ours) + A.P. (ours)	**735.9**

(b) Preparation: Task-Free Anticipatory Planning

TABLE I: Average cost per task in a 10-task sequence, averaged over 100 sequences in each of 32 Blockworld environments.

For each environment, we evaluate over 100, 10-length task sequences, drawn according to the task distribution $\tau \sim P(\tau)$, each in 32 different randomly generated environments, for a total of 32,000 task executions. For each trial, we evaluate performance using four planning approaches that leverage different aspects of our learning-augmented anticipatory planning approach:

Non-Learned Myopic Baseline (N.L. Myopic) Classical planning via Fast Downward [45], without any anticipatory planning cost. Plans only to minimize immediate cost $V^*_\tau(s_0)$.

Anticipatory Planning (A.P.) Planning is augmented with estimates of the anticipatory planning cost. Planning seeks to reduce the overall cost of both the immediate task and a single future task, Eq. (2).

Preparation + Myopic Baseline (Prep + N.L. Myopic) Non-learned myopic planning, yet the robot is first allowed to *prepare* the environment, Eq. (4).

Preparation + Anticipatory Planning (Prep + A.P.) Anticipatory planning, yet the robot is first allowed to *prepare* the environment, Eq. (4).

We show our results in Table I, which show that anticipatory planning `A.P.` (in the absence of preparation) reduces overall planning cost across all sequences by 5%. `N.L. Myopic` has higher planning costs because the plans generated from the task will introduce side effects on subsequent tasks, increasing the cost of their plans.

We additionally evaluate scenarios in which we first allow both robots to *prepare* their environment via task-free anticipatory planning, Eq. (4). When allowed to prepare the environment in advance using estimates of the anticipatory cost $V^*_{\text{A.P.}}$, the performance of both planners improves. After preparation, even the non-learned baseline `Prep+N.L. Myopic` shows improved performance by 5%, since objects that could potentially clutter and obstruct its plans are moved out of the way in advance. With preparation, our anticipatory planning approach `Prep+A.P.` improves overall performance by 11% compared to the non-prepared, non-learned baseline `N.L. Myopic`.

Critically, even in the absence of preparation, the task cost for our anticipatory planning approach (`A.P.`) gradually decreases as it accomplishes more tasks, suggesting that

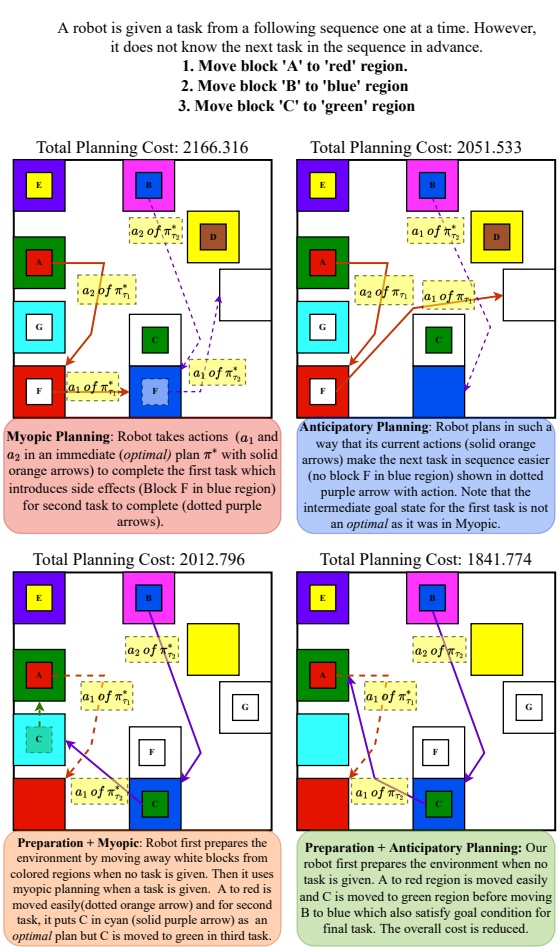


Fig. 5: **Anticipatory Planning Reduces Total Planning Cost of a Sequence:** We provide the robot to solve the sequence of three tasks. We see the total planning cost is improved with anticipatory planning. The robot minimizes the overall cost according to Eq. (2) in which it estimates $V^*_{A.P.}$.

it is gradually *preparing* or *organizing* the environment, incrementally making it cheaper to plan. By the end of the sequences (at `T10` in Fig. 6), the average task cost approaches that of the initially prepared environment with anticipatory planning.

Furthermore, to qualitatively illustrate the benefits of both anticipatory task planning and preparation, we highlight one result scenario in Fig. 5. An example task sequence is given to the robot to complete one task at a time: *[(`T1`) Move block `A` to the `red` region, (`T2`) Move block `B` to the `blue` region, (`T3`) Move block `C` to the `green` region].* `N.L. Myopic` first places block `F` in the `blue` region and block `A` in the `red` region to satisfy the goal condition. However, its placement of block `F` on the `blue` region introduces a side effect, so that the robot must subsequently take an additional step to complete the second task: moving block `F` to the `white` region. Under our anticipatory planning approach, the white block `F` is moved directly to the `white` region, avoiding this expensive side effect at the expense of a small initial cost.

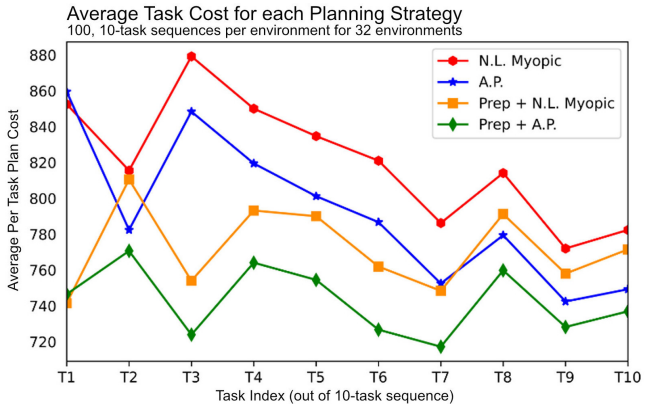


Fig. 6: Average total cost of planning 100, 10-task sequences in each of 32 unique Blockworld Environments. Each task sequence is unique and randomly sampled from the set of task; the "task index" thus corresponds to how many tasks in each sequence have so far been completed.

VIII. Conclusion and Future Work

In this work, we present Anticipatory Planning, an approach for learning-augmented planning that aims to reduce planning costs for long-lived household robots expected to accomplish sequences of tasks, yet given only one at a time. Through the estimation of the *anticipatory planning cost*, and thus the expected impact actions on potential future actions, our approach guides the robot towards behaviors that reduce overall planning cost.

Additionally, when a task is not given—a common scenario for household robots, which may not be constantly given active instruction—the robot is allowed to *prepare* the environment (task free anticipatory planning) so as to reduce expected future costs when it is eventually given a task.

In future work, we seek to apply this approach to guide *integrated* Task and Motion Planning, which would make our approach more broadly applicable in all manner of household domains in which the robot's implicit decisions can negatively impact future tasks and the state of its surroundings. In such environments, one could use an existing dataset, such as the ALFRED task benchmark [50], to provide tasks to define the task distribution.

Acknowledgement

We thank the Robust Robotics Group at MIT, Leslie Kaelbling, Nicholas Roy, Jana Kosecka, George Konidaris, Tom Silver, and Rohan Chitnis for their thoughtful feedback on this work. We acknowledge funding support from George Mason University.

References

[1] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, pp. 61–124, 2003.

[2] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 440–448.

[3] E. Plaku and G. D. Hager, "Sampling-based motion and symbolic action planning with geometric and differential constraints," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 5002–5008.

[4] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[5] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now. in 2011 ieee icra," 2011.

[6] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 639–646.

[7] B. Kim, Z. Wang, L. P. Kaelbling, and T. Lozano-Pérez, "Learning to guide task and motion planning using score-space representation," *The International Journal of Robotics Research*, vol. 38, no. 7, pp. 793–812, 2019.

[8] R. Chitnis, D. Hadfield-Menell, A. Gupta, S. Srivastava, E. Groshev, C. Lin, and P. Abbeel, "Guided search for task and motion plans using learned heuristics," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 447–454.

[9] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Robotics: Science and systems*, vol. 12. Ann Arbor, MI, USA, 2016, p. 00052.

[10] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[11] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[13] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[14] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, "Active model learning and diverse action sampling for task and motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4107–4114.

[15] R. Chitnis, L. P. Kaelbling, and T. Lozano-Pérez, "Learning quickly to plan quickly using modular meta-learning," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 7865–7871.

[16] B. Kim and L. Shimanuki, "Learning value functions with relational state representations for guiding task-and-motion planning," in *Conference on Robot Learning*. PMLR, 2020, pp. 955–968.

[17] P. Agrawal, "The task specification problem," in *Conference on Robot Learning (CoRL) Blue Sky Submission Track*, 2021.

[18] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning (ICML)*, 2017.

[19] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International Conference on Machine Learning (ICML)*, 2020, pp. 1597–1607.

[20] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," *arXiv preprint arXiv:1802.06070*, 2018.

[21] D. Kim, "Imitation learning for sequential manipulation tasks: Leveraging language and perception," Master's thesis, Massachusetts Institute of Technology, 2021.

[22] H. J. Jeon, S. Milli, and A. Dragan, "Reward-rational (implicit) choice: A unifying formalism for reward learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 4415–4426.

[23] R. Shah, P. Freire, N. Alex, R. Freedman, D. Krasheninnikov, L. Chan, M. D. Dennis, P. Abbeel, A. Dragan, and S. Russell, "Benefits of assistance over reward learning," 2021. [Online]. Available: https://openreview.net/forum?id=DFIoGDZejIB

[24] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 29, 2016.

[25] H. Mahzoon, Y. Yoshikawa, and H. Ishiguro, "Ostensive-cue sensitive learning and exclusive evaluation of policies: A solution for measuring contingency of experiences for social developmental robot," *Frontiers in Robotics and AI*, vol. 6, p. 2, 2019.

[26] R. Choudhury, G. Swamy, D. Hadfield-Menell, and A. D. Dragan, "On the utility of model learning in hri," in *International Conference on Human-Robot Interaction (HRI)*, 2019, pp. 317–325.

[27] L. Sun, W. Zhan, M. Tomizuka, and A. D. Dragan, "Courteous autonomous cars," in *International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 663–670.

[28] R. Shah, D. Krasheninnikov, J. Alexander, P. Abbeel, and A. Dragan, "Preferences implicit in the state of the world," *arXiv preprint arXiv:1902.04198*, 2019.

[29] V. Krakovna, L. Orseau, R. Ngo, M. Martic, and S. Legg, "Avoiding side effects by considering future tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 064–19 074, 2020.

[30] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 297–330, 2020.

[31] D. Whitney, E. Rosen, and S. Tellex, "Learning from crowdsourced virtual reality demonstrations," in *International Workshop on Virtual, Augmented, and Mixed Reality for HRI (VAM-HRI)*, 2018.

[32] C. Moro, G. Nejat, and A. Mihailidis, "Learning and personalizing socially assistive robot behaviors to aid with activities of daily living," *ACM Transactions on Human-Robot Interaction (THRI)*, vol. 7, no. 2, pp. 1–25, 2018.

[33] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal, "Skill learning and task outcome prediction for manipulation," in *International Conference on Robotics and Automation (ICRA)*, 2011, pp. 3828–3834.

[34] J. van den Berg, S. Miller, D. Duckworth, H. Hu, A. Wan, X.-Y. Fu, K. Goldberg, and P. Abbeel, "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations," in *International Conference on Robotics and Automation (ICRA)*, 2010, pp. 2074–2081.

[35] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning to sequence movement primitives from demonstrations," in *International Conference on Intelligent Robots and Systems (IJRR)*, 2014, pp. 4414–4421.

[36] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *The International Journal of Robotics Research (IJRR)*, vol. 31, no. 3, pp. 360–375, 2012.

[37] M. Li, A. Canberk, D. P. Losey, and D. Sadigh, "Learning human objectives from sequences of physical corrections," in *International Conference on Robotics and Automation (ICRA)*, 2021, pp. 2877–2883.

[38] A. Mészáros, G. Franzese, and J. Kober, "Learning to pick at non-zero-velocity from interactive demonstrations," *Robotics and Automation Letters (RA-L)*, vol. 7, no. 3, pp. 6052–6059, 2022.

[39] S. Choi, K. Lee, S. Lim, and S. Oh, "Uncertainty-aware learning from demonstration using mixture density networks with sampling-free variance modeling," in *International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6915–6922.

[40] S. Thakur, H. van Hoof, J. C. G. Higuera, D. Precup, and D. Meger, "Uncertainty aware learning from demonstrations in multiple contexts using bayesian neural networks," in *International Conference on Robotics and Automation (ICRA)*, 2019, pp. 768–774.

[41] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[42] T. Silver, R. Chitnis, A. Curtis, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Planning with learned object importance in large problem instances using graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 13, 2021, pp. 11 962–11 971.

[43] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *International Conference on Robotics and Automation (ICRA)*, vol. 1, 2000, pp. 521–528.

[44] 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.
[45] M. Helmert, "The Fast Downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.
[46] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *J. Artif. Int. Res.*, vol. 14, no. 1, p. 253–302, May 2001.
[47] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.
[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[49] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.
[50] M. Shridhar, J. Thomason, D. Gordon, Y. Bisk, W. Han, R. Mottaghi, L. Zettlemoyer, and D. Fox, "Alfred: A benchmark for interpreting grounded instructions for everyday tasks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 740–10 749.