

Effort Allocation for Deadline-Aware Task and Motion Planning: A Metareasoning Approach

Yoonchang Sung*, Shahaf S. Shperberg*, Qi Wang*, and Peter Stone

Abstract

In robot planning, tasks can often be achieved through multiple options, each consisting of several actions. This work specifically addresses deadline constraints in task and motion planning, aiming to find a plan that can be executed within the deadline despite uncertain planning and execution times. We propose an effort allocation problem, formulated as a Markov decision process (MDP), to find such a plan by leveraging metareasoning perspectives to allocate computational resources among the given options. We formally prove the NP-hardness of the problem by reducing it from the knapsack problem.

Both a model-based approach, where transition models are learned from past experience, and a model-free approach, which overcomes the unavailability of prior data acquisition through reinforcement learning, are explored. For the model-based approach, we investigate Monte Carlo tree search (MCTS) to approximately solve the proposed MDP and further design heuristic schemes to tackle NP-hardness, leading to the approximate yet efficient algorithm called DP_Rerun. In experiments, DP_Rerun demonstrates promising performance comparable to MCTS while requiring negligible computation time.

Index Terms

Metareasoning, task and motion planning, resource-constrained planning.

Y. Sung is with the Department of Computer Science, The University of Texas at Austin, Austin, TX 78712 USA (e-mail:yoons8@cs.utexas.edu).

S. Shperberg is with the Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Be'er Sheva, Israel (e-mail:shperbsh@bgu.ac.il).

Q. Wang is with the Department of Computer Science, The University of Texas at Austin, Austin, TX 78712 USA (e-mail:harrywang@utexas.edu).

P. Stone is with the Department of Computer Science, The University of Texas at Austin, Austin, TX 78712 USA and Sony AI (e-mail:pstone@cs.utexas.edu).

*Equal contribution.

I. INTRODUCTION

Imagine a scenario where a bus leaves from the station in 30 minutes. There are multiple routes to the station, but it is not known exactly how long each route will take, and even the computation times for determining the exact motions needed to execute each route are unknown a priori. Similarly, imagine you are a chef and need to serve a meal in 15 minutes, which requires a pot. There is a nearby pot at the bottom of a large stack of dishes such that retrieving it involves taking out several other objects, while taking out another pot located far away does not involve taking out any other objects. In such scenarios, we need to reason about how long each option will take to execute, combined with how long it will take to even find an executable plan.

Even without deadlines, the above planning problems are challenging for robots, particularly when the planning horizon is long, as finding a solution generally involves searching in high dimensions with significant depth and branching factors. Task and motion planning (TAMP) introduces abstractions—such as passing through a particular intersection in the bus example and picking up a dish in the meal preparation example—defined at the symbolic level. These abstractions do not specify how the robot can realize low-level motions (*i.e.*, trajectories or motor commands) to achieve these abstract actions, allowing for planning in the abstract space and avoiding the search complexity of the low-level motion space. However, such planning is generally done without full knowledge of how long it will take to translate an abstract action into a fully executable sequence of low-level commands.

Introducing the deadline constraint to TAMP problems induces an additional problem of scheduling computation allocation among candidate abstract plans. An abstract plan, referred to as a plan skeleton, corresponds to a sequence of abstract actions. The TAMP problem and computation selection may be interleaved, with computation selection determining which abstract plan the planner must consider and affecting the probability of finding a solution within a deadline. Essentially, computation selection can be viewed as metareasoning to effectively solve deadline-constrained TAMP problems.

In this work, we introduce a Markov decision process (MDP) formulation that involves metareasoning for computation selection in deadline-constrained TAMP problems. The introduced problem is inherently stochastic, as exact motions are unknown a priori before computation,

resulting in stochastic planning and execution times. To address this stochasticity, we explore both model-based and model-free approaches. The model-based approach involves learning transition models when data on past planning experience is available, while the model-free approach is used when such data is unavailable.

We additionally show that the proposed metareasoning MDP problem is NP-hard, proven by reduction from the knapsack problem. To alleviate the complexity of the problem, we employ the Monte Carlo tree search (MCTS) scheme to approximately solve the problem and further propose heuristics that exploit the structure of an optimal policy, which can be solved in polynomial time at the expense of optimality. However, in experiments, we show that the problem restricted by heuristics yields solutions that are comparable to those from the original problem while greatly improving efficiency.

Our main contributions can be summarized as follows:

- We propose a deadline-aware TAMP formulation that leverages metareasoning perspectives. To the best of our knowledge, our work is the first to address time-critical scenarios in TAMP.
- We formally prove the NP-hardness of the proposed metareasoning problem and design a polynomial-time approximate algorithm, called DP_Rerun, through heuristics.
- Furthermore, we investigate approximately solving the metareasoning problem using MCTS and employing reinforcement learning when data is not available a priori.
- We test both navigation and manipulation scenarios to evaluate the effectiveness of the proposed method, DP_Rerun, in practice. We find empirically that it significantly outperforms baselines in experiments. In particular, DP_Rerun takes significantly less computation time to achieve performance almost on par with MCTS.

The rest of this paper is organized as follows. Section II introduces related work on TAMP and metareasoning. Section III presents our metareasoning formulation as an MDP problem. The hardness of the proposed problem is derived in Section IV. Model-based and model-free solution methods are introduced in Sections V and VI, respectively. Section VII illustrates the practical use cases of the proposed method in navigation and manipulation domains. Section VIII presents the experimental results, and concluding remarks are included in Section IX.

II. RELATED WORK

Here, we review the related literature on task and motion planning, resource-constrained planning, and metareasoning to highlight the overlooked challenges addressed in this work.

A. Task and motion planning

TAMP problems involve manipulating multiple objects in the world, requiring reasoning about where to grasp an object, where to place it, and robot motion commands [1, 2]. TAMP planners introduce symbolic abstractions that reason primarily about objects to separate out low-level motion planning. Essentially, TAMP exhibits a bilevel structure: high-level task reasoning (*i.e.*, determining which actions to take) and low-level motion reasoning (*i.e.*, determining how to execute these actions) mutually provide complementary guidance.

Existing TAMP planners can be generally classified into three approaches [1]. The *satisfy-before-sequence* approach [3, 4, 5] involves finding a valid assignment of values to continuous variables that satisfy the corresponding constraints. Subsequently, the sampled values are sequenced to generate a complete plan executable by the robot. On the other hand, the *sequence-before-satisfy* approach [6, 7, 8, 9, 10, 11, 12, 13] reverses the search process by first computing abstract plans that do not involve reasoning about continuous variables, followed by finding satisfying values for those variables. An alternative approach is *interleaved-satisfy-and-sequence* [14, 15, 16, 17], whereby a search tree is constructed to jointly search for satisfying values for both discrete and continuous variables in an interleaved manner.

Two main directions have been actively pursued in the community to advance the state of the art in TAMP research: (1) focusing on improving planning efficiency and (2) developing a learning framework to acquire models for TAMP planners from data, rather than relying solely on hand-designed models. Several ideas have been proposed to enhance planning efficiency, such as novel heuristics for informed search control [5, 18, 13], learning samplers for effectively handling continuous variables [19, 20], value function learning to estimate the contribution to reaching a goal [16], and feasibility prediction to mitigate the failure of expensive motion planning [21, 22, 23, 24, 25]. TAMP planners require models, such as symbolic models and skills, to effectively find a solution. Data-driven model learning efforts for TAMP include skill learning [26, 27, 28, 29, 30], world model learning [31, 32], and predicate learning [33].

As such, most TAMP research has focused on efficiently and effectively finding a satisfying plan, while little has addressed optimal planning [34, 35, 36]. To the best of our knowledge, no previous work has tackled deadline-aware planning in TAMP. This work introduces the first deadline-aware TAMP formulation, highlighting the distinctive challenges posed by stochastic planning and execution times, as well as the allocation of computation among available options.

B. *Resource-constrained planning*

In robot planning, it is often essential to consider resource constraints to adhere to limited time or energy budgets, given that a robot’s operational lifetime is finite. Various forms of these constraints have been explored in the literature [37]; here, we only discuss representative classes of problems.

Prize collecting traveling salesman problem [38] is a variant of the well-known traveling salesman problem. Its goal is to visit a subset of vertices from a given graph in a way that minimizes both the traveling distance and the net penalties associated with collecting prizes and penalties. Another related class of problems is the *orienteering problem* [39, 40, 41], whose objective is to maximize the reward collected until a given budget is exhausted. Various algorithms have been developed, including the recursive greedy algorithm [39], which offers a provable approximate guarantee. *Multi-armed bandit* problems [40] involve maximizing the expected return when dealing with a finite number of arms associated with unknown black-box reward functions. When only a fixed number of trials is allowed, resource constraints are considered, making the exploration-exploitation dilemma critical.

Although both the problems introduced in this subsection and the problem proposed in this work address bounded resources, the introduced problem includes a unique structure that makes it distinctive, thereby necessitating the consideration of metareasoning. Specifically, the problem involves two interconnected subproblems: one is the original problem of solving a given TAMP problem, and the other is the metareasoning problem of scheduling computation allocations among options to meet a deadline. We provide more detailed explanations on the connection to metareasoning in the following subsection.

C. Metareasoning

Rational metareasoning [42] has been investigated in the context of developing intelligent agents under constraints of bounded resources, with two early proposed equivalent models: anytime algorithms [43] and flexible computation [44]. In this subsection, our focus is on the planning-related literature. Several more comprehensive surveys can be found in the literature [45, 46, 47, 48].

The *value of computation* is a critical concept in metareasoning, calculated as the improvement of solution quality resulting from a computation, subtracted by the computationally incurred costs [44]. Since this value is difficult to compute, how to approximate the value of computation has become a subject of research. Moreover, the improvement of solution quality in reality is often uncertain. Therefore, monitoring and control schemes have been proposed to effectively handle stochastic solution quality improvement [49, 50, 51, 52].

One constructive application of metareasoning for planning is to determine the optimal moment to stop planning and begin executing a computed plan through interleaved planning and execution [53, 54]. This approach is particularly valuable when the reasoning procedure exhibits anytime behavior, allowing the agent to achieve a higher-quality plan by investing more time in planning despite incurring greater costs, such as energy consumption. Therefore, monitoring the progress of plan quality improvement is crucial for establishing a stopping policy that effectively balances solution quality and computation time.

Metareasoning has also been employed in robotics problems, such as cost-effectively stopping for optimal motion planning [55] and handling exceptional situations to ensure the safety of autonomous driving [56].

Another related constructive use case is to select which computation to utilize when multiple computations are available under limited computational resources. Metareasoning has been extensively applied to find the optimal computation among various alternatives. Hay *et al.* [57] adopted metareasoning in the selection phase of MCTS to determine which future sequences to simulate and compared it with bandit settings. Lieder *et al.* [58] developed a theory for algorithm selection by metareasoning to model human strategy selection. Callaway *et al.* [59] proposed a learning algorithm for approximating the optimal selection of computations.

Our work also involves computation selection considered as metareasoning. However, there

are two notable features in our work that distinguish it from the aforementioned literature. First, we address deadline-aware planning, meaning that the total duration of selected sequential computations must meet a deadline, whereas existing work treats each selected computation as a separate time-critical problem. This fundamental difference prevents the proposed work from utilizing the conventional notion of the value of computation. Second, our work considers TAMP as an object-level problem, which has not yet been addressed in the literature.

The most related work to ours is by Shperberg *et al.* [60], where they explored a metareasoning problem within the domain of situated temporal (symbolic) planning [61], a planning paradigm that considers the time elapsed during plan search while also accounting for a pre-specified deadline. They abstracted the problem of searching for plans into a meta-level scheduling problem, bypassing the complexities of plan state representation and search procedures. Their approach involved modeling the problem using a set of processes, each dedicated to searching for a plan, akin to representing search nodes on an open list. Each process is characterized by a probabilistic performance profile, modeled by a random variable indicating the probability of successful termination given processing time, as well as a random variable modeling the deadline corresponding to each partial plan, which is only revealed after planning is concluded. The meta-level problem lies in finding an optimal schedule of processing time across all processes that maximizes the probability that any process delivers a plan before its deadline. A simplified version of this problem, known as “simplified allocating planning effort when actions expire,” assumes discrete time intervals and has been proven to be NP-hard. However, under the condition of known deadlines, the problem becomes solvable in pseudo-polynomial time through dynamic programming. Later, this line of work was extended to consider interleaved planning and execution, where partial plans can be executed during the search [62, 63]. While this body of work bears relevance to our research, it primarily concentrates on deriving symbolic plans. In contrast, our focus lies in elaborating existing symbolic plans through motion-level reasoning to make them executable for a robot, optimizing the likelihood of meeting a pre-specified deadline.

III. PROBLEM FORMULATION

Consider a robot that can interact with objects in the world. Its *configuration* space is represented by \mathbb{Q} , and its space of *grasps* is represented by \mathbb{G} , corresponding to a space of the 6D pose of the end-effector in $SE(3)$, the grasp preshape, and the approach direction [64].

We assume that the world is composed of a finite set of objects $\mathcal{O} = \mathcal{O}_F \cup \mathcal{O}_M$, where *fixed objects* \mathcal{O}_F include objects such as floors, walls, tables, and shelves, and *movable objects* \mathcal{O}_M include objects such as cups, books, and keys that are movable by the robot. A subset of fixed objects \mathcal{O}_F , such as tables and shelves, is endowed with *workspace regions* in \mathbb{R}^3 , where movable objects can be placed. The space of *poses* for movable object i in \mathcal{O}_M is denoted by \mathbb{P}_i , representing stable placements in the workspace regions.

The *composite state space* of the robot and objects is represented by $\mathbb{Q} \times \mathbb{G} \times \prod_{i=1}^M \mathbb{P}_i$, where M denotes the number of movable objects. Each variable corresponding to a state space component is referred to as a *typed variable*, which includes a robot configuration $q \in \mathbb{Q}$, a grasp pose $g \in \mathbb{G}$, and the pose $p_i \in \mathbb{P}_i$ of each movable object i . The *state* is then a tuple of continuous values assigned to the concatenation of these typed variables.

We assume deterministic transitions as a result of actions and error-free perception. Despite these restrictive assumptions, TAMP planning remains computationally intractable, as the underlying problems of task planning [65] and motion planning [66, 67] have each been proven to be PSPACE-complete. Our aim is to establish an initial solution as foundational for future efforts aimed at relaxing these assumptions. However, we introduce novel challenges regarding imperfect knowledge in this work, where planning time and execution time of an action are unknown, which are crucial for addressing deadline-aware planning.

We now recap a general TAMP formulation [1], a planning framework that finds a sequence of actions to achieve a certain goal, without considering deadlines. We then broaden the problem description to include effort allocation for finding a TAMP solution that can be planned and executed within a pre-specified deadline.

A. TAMP formulation

The TAMP formulation often leverages a logic-based action language, such as planning domain definition language (PDDL [68]), to enable high-level symbolic planning for efficient low-level motion planning [12]. We define a TAMP problem as a tuple $\langle \mathcal{O}, \mathcal{P}, \mathcal{I}, \mathcal{G}, \Delta \rangle$ as follows:

- \mathcal{O} denotes a finite set of objects introduced previously.
- \mathcal{P} denotes a finite set of *predicates*, where each predicate is a Boolean function that can be evaluated on a tuple of object variables $o \in \mathcal{O}$ and typed variables to determine whether it is true or false. An assignment of values to a predicate is called a *literal*. For example,

$\text{InRegion}(o_i \in \mathcal{O}_M, o_f \in \mathcal{O}_F, p_i \in \mathbb{P}_i)$ is true if the movable object o_i with pose p_i is in the workspace of the fixed object o_f , $\text{InHand}(o_i \in \mathcal{O}_M, g \in \mathbb{G})$ is true if the movable object o_i is stably grasped by the robot with grasp pose g , and $\text{Reachable}(q \in \mathbb{Q}, q' \in \mathbb{Q})$ is true if there exists a collision-free path between configurations q and q' . Notice that the motion planning aspect is addressed through the evaluation of predicates containing typed variables, while task planning deals with discrete variables representing objects.

- \mathcal{I} denotes a set of initial literals. For ease of presentation, we also refer to a tuple of literals and the state of typed variables as a state. Therefore, \mathcal{I} along with assigned values of typed variables describe the initial state of the world.
- \mathcal{G} denotes a conjunctive set of goal literals.
- Δ denotes a finite set of *actions*, whose arguments are tuples of object variables and typed variables. Each action $\delta \in \Delta$ is described by a set of literal *preconditions* $\text{pre}(\delta)$ and a set of literal *effects* $\text{eff}(\delta)$. We use $+$ and $-$ as superscripts on pre and eff to represent positive and negative literals, respectively. An action with an assignment of values is applicable to a state if $\text{pre}^+(\delta) \subseteq \mathcal{I}$ and $\text{pre}^-(\delta) \cap \mathcal{I} = \emptyset$. The successor state becomes a tuple of $\mathcal{I} \setminus \text{eff}^-(\delta) \cup \text{eff}^+(\delta)$ along with the values of typed variables assigned in the action.

A solution to the TAMP problem is a finite sequence of actions from the initial state, specified by the initial literals \mathcal{I} , to reach the goal state, satisfying the goal literals \mathcal{G} . A partial plan, where object variables in a sequence of actions in the plan are only determined (often called *grounding*) while continuous typed variables are not yet chosen (often called *refinement*), is referred to as a *plan skeleton* [69]. Predicates involving only typed variables, such as $\text{Reachable}(q \in \mathbb{Q}, q' \in \mathbb{Q})$, are always considered true when finding plan skeletons.

Refinement of actions in a plan skeleton, involving the evaluation of $\text{Reachable}(q \in \mathbb{Q}, q' \in \mathbb{Q})$ predicates, finds valid robot motions that satisfy geometric and kinematic constraints, such as arm motion for grasping a cup and base motion to reach a table. Geometric constraints ensure that the robot does not collide with any objects in the world or itself at any time, and that no collision occurs among any pairs of objects. Kinematic constraints govern the degrees of freedom of the robot and the relationships among links connected through joints.

A finite sequence of valid motions obtained by refinement constitutes a continuous path in \mathbb{Q} that the robot follows from its initial state to reach a goal. Given a specification of the robot's motion model, execution time can be computed from the resultant path. Optimal planners aim

at finding a plan that minimizes the total execution time [34, 35, 36]. However, in this work, our goal is to identify any plan that can be executed before a deadline, while also taking into account the planning (or computation) time required to identify it.

In Section VII, we provide example scenarios in navigation and manipulation domains to demonstrate TAMP problem specifications in practice.

B. Effort allocation for deadline-aware TAMP problems

The TAMP problems introduced above do not take into account the notion of a deadline or time budget, so their solvers only aim at finding a satisfying plan regardless of the time it takes to find it. However, in this work, we explicitly address deadline-aware planning, which imposes an additional constraint ensuring that a plan found must be refinable and executable within a deadline.

Our work adheres to the *sequence-before-satisfy* approach [6, 7, 8, 9, 10, 11, 12, 13], often leveraging AI heuristics, such as Fast Downward [70], to efficiently find plan skeletons initially without considering continuous typed variables. The search for plan skeletons is then followed by finding satisfying values for the typed variables.

Since plan skeleton search is generally computationally tractable compared to continuous value selection, one can first find K plan skeletons to jointly search for continuous values satisfying constraints, a process referred to as top- K planning [71, 72]. This approach may lead to finding a solution quickly, as any value selection in some of the plan skeletons may violate the constraints, and thus, not yield solutions. Top- K plan skeletons can be chosen by introducing either a user-defined cost or unit cost on every action. In the latter case, K plan skeletons are selected starting from the smallest number of actions reaching the goal in ascending order.

In this work, we consider a scenario where multiple plan skeletons are provided, but the planning (or refining) time and execution time of individual actions within the plan skeletons are *unknown*. It is important to note that we interchangeably use (motion) “planning” and “elaboration” to mean refinement throughout the paper. Let $\Sigma = \{\sigma_1, \dots, \sigma_k, \dots, \sigma_K\}$ be a finite set of top- K plan skeletons. Each plan skeleton σ_k contains A_k actions, denoted by $\sigma_k = (\delta_k^1, \dots, \delta_k^j, \dots, \delta_k^{A_k})$, where δ_k^j denotes the j -th action in plan skeleton σ_k .

We consider *discrete* time steps for both planning time and execution time, where the interval between sequential time steps is determined by a user. Let $\{1, \dots, D\}$ denote a set of decision-

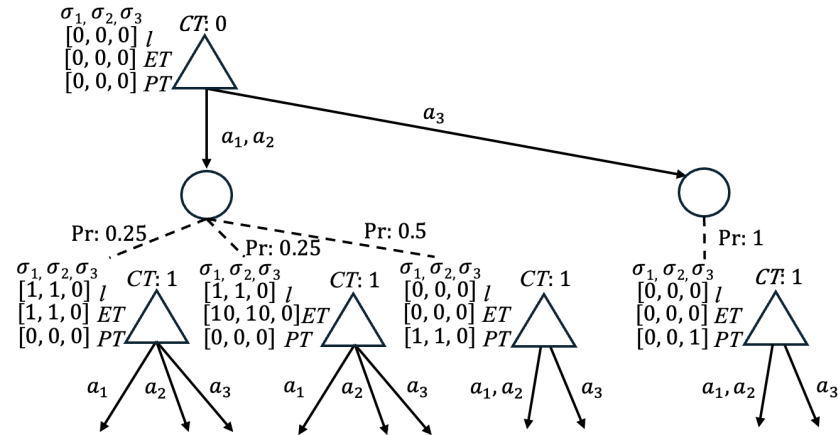
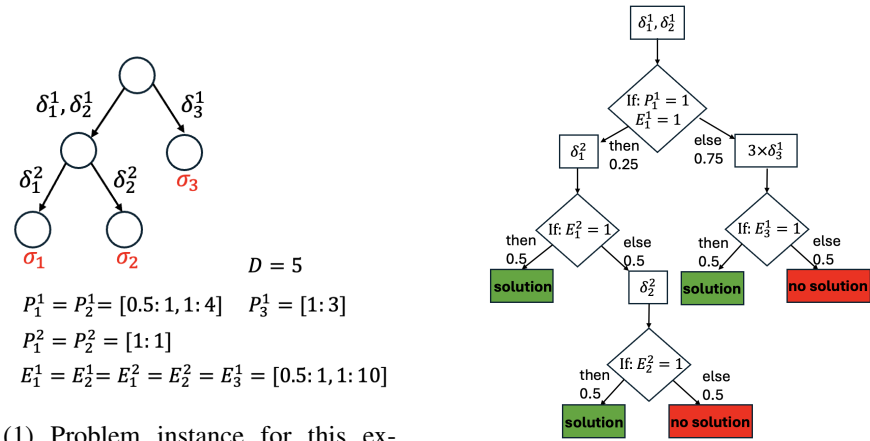


Fig. 1: Example of an effort allocation problem.

making time steps, where the planner decides which plan skeleton to allocate one interval of its time (or effort) to refine a corresponding action, that is, finding satisfying continuous values of typed variables of the action. Multiplying a user-defined time interval by D yields the total wall-clock time allotted to the planner, corresponding to a *deadline*.

To handle unknown planning time and execution time, we propose a learning paradigm for estimating these quantities using collected data obtained from past experiences. We round both the planning time and execution time of every action to match with decision-making time steps and introduce discrete distribution functions to model the uncertainty about these quantities. Specifically, we define P_k^j , a discrete cumulative distribution function (CDF) modeling a *planning time distribution* required to find valid continuous values of typed variables that satisfy corresponding constraints, and E_k^j a discrete CDF modeling an *execution time distribution* required for the robot to execute a motion obtained by the refined action. For convenience, we define p_k^j and e_k^j as the probability mass functions (PMF) corresponding to P_k^j and E_k^j , respectively. These PMFs can be computed directly from their corresponding CDFs.

Note that once the planner computes a motion, the execution time to execute the motion can be obtained deterministically as it depends on the path length. However, before a path is obtained, the execution time is stochastic. The stochasticity arises from the random process of generating motions in motion planning.

We finally propose an *effort allocation problem*, whose objective is to maximize the probability of finding a fully refined plan skeleton that can be planned and executed by the robot within a pre-specified deadline, under uncertain planning times and execution times, by learning a computation allocation policy among plan skeletons.

An example of effort allocation for a deadline-aware TAMP problem instance is illustrated in Figure 1 (1). In this example, there are three plan skeletons: $\sigma_1 = (\delta_1^1, \delta_1^2)$, $\sigma_2 = (\delta_2^1, \delta_2^2)$, and $\sigma_3 = (\delta_3^1)$. Note that σ_1 and σ_2 share the first action, *i.e.*, $\delta_1^1 = \delta_2^1$. The deadline D is 5 time steps. All actions have the same execution time probability mass function: they finish in 1 time step 50% of the time and in 10 time steps the other 50% of the time. Additionally, in this example, actions δ_1^2 , δ_2^2 , and δ_3^1 have deterministic planning durations: three steps for δ_3^1 and one step each for δ_1^2 and δ_2^2 . Action δ_1^1 (or δ_2^1) has a stochastic planning duration, taking either one or four time steps with equal probability.

The optimal policy in this scenario, depicted in Figure 1 (2) is to first execute the motion

planner on δ_1^1 (or δ_2^1) for 1 time step. If motion planning has not found a plan, or if it found a plan requiring 10 time steps for execution, the motion planner then proceeds to refine δ_3^1 for three time steps. This refinement is guaranteed to succeed; however, the resulting plan can be executed on time only if the execution time is 1 step (not 10); otherwise, no solution is found. If δ_1^1 (or δ_2^1) completes planning within 1 time step and discovers a motion plan executable in 1 time step, the motion planner continues to refine δ_1^2 for 1 time step. If this plan can be executed in 1 time step, a solution is found. Otherwise, the motion planner proceeds to refine δ_2^2 for 1 time step. If the resulting plan can be executed in 1 time step, a solution is found; otherwise, no solution is found. The overall success probability of this policy is 0.5625. This value can be extracted from the figure by multiplying the probabilities along each path leading to a leaf node where a solution is found and then summing the probabilities across these different paths.

In the following subsection, we finalize our problem formulation using the notation introduced so far.

C. An MDP model

We formalize the effort allocation problem using a Markov decision process (MDP) model, which can be represented by a tuple $\langle \mathcal{S}, \mathcal{A}, T, R, D \rangle$ as follows:

- \mathcal{S} denotes a finite set of MDP states, where $\mathcal{S} = \{(CT, l_1, \dots, l_k, \dots, l_K, PT_1, \dots, PT_k, \dots, PT_K, ET_1, \dots, ET_k, \dots, ET_K), \text{success}, \text{failure}\}$. Here, $CT \in \mathbb{N}$ denotes the current time step. $l_k \in \{0, \dots, A_k\}$ is an index of the last action in plan skeleton σ_k that has already been refined to find a path, where $l_k = 0$ indicates that no actions in the plan skeleton have been refined yet. $PT_k \in \mathbb{N}$ denotes accumulated time steps spent for planning so far for action $\delta_k^{l_k+1}$ in plan skeleton σ_k . PT_k maintains accumulated time steps for action $\delta_k^{l_k+1}$ specifically instead of action $\delta_k^{l_k}$ because the planner only needs to keep track of planning time for *unrefined* actions. For actions that are refined, the planner has already confirmed the actual planning times, so refined actions no longer have stochasticity. $ET_k \in \mathbb{N}$ denotes accumulated time steps that will be spent for execution for all actions refined so far in plan skeleton σ_k . *success* and *failure* denote the terminal states, where *success* implies the successful refinement of one of the K plan skeletons before the deadline, and *failure* implies the opposite.

- $\mathcal{A} = \{a_1, \dots, a_k, \dots, a_K\}$ denotes a finite set of MDP actions, where MDP action a_k selects plan skeleton σ_k to spend one time step refining action $\delta_k^{l_k+1}$.
- T is a transition function, where $T(s \in \mathcal{S}, a_k \in \mathcal{A}, s' \in \mathcal{S})$ represents the probability of transitioning from MDP state $s = (CT, l_1, \dots, l_k, \dots, l_K, PT_1, \dots, PT_k, \dots, PT_K, ET_1, \dots, ET_k, \dots, ET_K)$ to MDP successor state $s' = (CT', l'_1, \dots, l'_k, \dots, l'_K, PT'_1, \dots, PT'_k, \dots, PT'_K, ET'_1, \dots, ET'_k, \dots, ET'_K)$ by taking MDP action a_k . We define four types of transitions as follows:
 - 1) $CT' = CT + 1$ with a probability of 1. We add an additional time step for each decision-making time step.
 - 2) $l'_m, PT'_m, ET'_m = l_m, PT_m, ET_m$ with a probability of 1 for all m such that $\delta_m^{l_m+1} \neq \delta_k^{l_k+1}$.
 - 3) For action $\delta_k^{l_k+1}$, a path can be found with a probability of $P_k^{l_k+1}(PT_k + 1)$. For all m such that $\delta_m^{l_m+1} = \delta_k^{l_k+1}$, if $l_m < A_k$, then $l'_m, PT'_m, ET'_m = l_m + 1, 0, ET_m + x$, where $x \in \text{support}(e_k^{l_k+1})^1$, with a probability of $P_k^{l_k+1}(PT_k + 1) \cdot e_k^{l_k+1}(x)$, or success otherwise.
 - 4) Alternatively, a path cannot be found with a probability of $1 - P_k^{l_k+1}(PT_k + 1)$ even after spending the current time step. In such a case, for all m such that $\delta_m^{l_m+1} = \delta_k^{l_k+1}$, if $CT < D$, then $l'_m, PT'_m, ET'_m = l_m, PT_m + 1, ET_m$, or failure otherwise.
- R is a reward function. The reward is 1 if the transition reaches the `success` terminal state, or 0 if it reaches the `failure` terminal state.

The objective of this MDP formulation is to find an optimal policy that maps each state from \mathcal{S} to an action from \mathcal{A} , leading to maximizing the expected cumulative reward.

It is important to note that our formulation does not require assumptions of *independent* plan skeletons or *downward refinability* [73]. Even if an MDP action chooses plan skeleton σ_k to refine a corresponding action $\delta_k^{l_k+1}$, for all m such that $m \neq k$ but $\delta_m^{l_m+1} = \delta_k^{l_k+1}$, the MDP updates the state for plan skeleton σ_m as well, thereby considering dependent plan skeletons. Downward refinability implies that plan skeletons are always refinable. However, our formulation can handle cases where a planning time distribution P_k^j does not always reach 1 even at the deadline, meaning that not all actions may be refinable. This capability effectively removes the

¹The `support` refers to the subset of the domain of a probability mass function E where the probability measure is non-zero.

downward refinability assumption and can address practical scenarios.

An example of the MDP is depicted in Figure 1 (3). The MDP is represented as an expectimax tree, where the initial state, with no time allocated to any plan skeleton, is the root. Only the first two levels of the tree are presented. From the root, the available actions are to either attempt to refine δ_1^1 or equivalently δ_2^1 by an MDP action a_1 or equivalently a_2 , or to try and refine δ_3^1 by an MDP action a_3 . If 1 time step is invested in refining δ_3^1 by an MDP action a_3 , the motion planner cannot find a plan, resulting in only one successor state where the planning time for σ_3 is increased by 1, with all other values remaining unchanged. If 1 time step is invested in refining δ_1^1 or equivalently δ_2^1 by an MDP action a_1 or equivalently a_2 , the motion planner does not finish planning with a probability of 0.5. In this case, the planning time for σ_1 and σ_2 is increased by 1 in the resulting state, with all other values unchanged. With a probability of 0.5, the refinement of action δ_1^1 or equivalently δ_2^1 is completed. Since the execution time of the refined plan can either be 1 or 10 (with equal probabilities), there are two possible resulting states. Both states will have $l_1 = 1$, $PT_1 = 0$, $l_2 = 1$, and $PT_2 = 0$, indicating that δ_1^1 or equivalently δ_2^1 has finished refinement. The first state has $ET_1 = ET_2 = 1$, while the second state has $ET_1 = ET_2 = 10$.

We also note that generalization of the proposed problem, making one plan skeleton more favorable than another even if both meet a deadline, is achievable by designing more complex MDP specifications. For instance, one could attach literals to MDP states and assign a reward to specific literals, such as reaching a meeting room before a deadline with a coffee grabbed, making that occurrence more favorable than merely making a meeting before the deadline.

The effort allocation problem reveals *metareasoning* perspectives as it allows efficient use of limited computational resources (*i.e.*, meta-level computation) for finding a solution to the TAMP problems (*i.e.*, object-level computation). Essentially, the effort allocation problem addresses when and what to consider, when to stop deliberation, and how to adaptively allocate computation under a limited computational budget (*i.e.*, deadline).

IV. THEORETICAL ANALYSIS

As shown in the previous section, the effort allocation problem can be formulated as an MDP. MDPs can be efficiently solved, for instance, using methods like value iteration, which have polynomial-time complexity with respect to the number of states and actions. However, the

challenge arises from the number of states in the MDP corresponding to all conceivable time allocations and their associated outcomes, which grows exponentially with the size of the input problem.

In this section, we establish that the effort allocation problem is NP-hard, even when the execution time is known. This finding suggests that addressing this problem necessitates the use of approximation algorithms or heuristic approaches to yield feasible solutions within reasonable computational bounds.

Theorem 1: Finding the optimal policy for the effort allocation problem is NP-hard.

Proof: We establish NP-hardness by reducing from the optimization version of the knapsack problem.

Definition 1 (Knapsack problem [74, problem MP9]): Given a finite set of items $\mathcal{S} = \{s_1, \dots, s_k, \dots, s_K\}$, each with a positive integer weight w_k and value v_k , and a weight limit W , find a subset \mathcal{S}° of \mathcal{S} such that the total weight of \mathcal{S}° is at most W and the total value of \mathcal{S}° is maximized.

Without loss of generality, we assume that all the values are greater than or equal to 1. That is, $v_k \geq 1$ for all $1 \leq k \leq K$. We also assume, without loss of generality, that $\mathcal{S}^\circ = \{s_1, \dots, s_m, \dots, s_M\}$ with $M \leq K$.

In our reduction, each plan skeleton represents an item in the knapsack problem. We only consider degenerate plan skeletons $\Sigma = \{\sigma_1, \dots, \sigma_k, \dots, \sigma_K\}$, where each plan skeleton includes only a single action, and actions do not overlap between plan skeletons. Formally, for all $1 \leq k \leq K$, $\sigma_k = (\delta_k^1)$ and $\delta_k^1 \neq \delta_m^1$ for all $k \neq m$. Additionally, we assign a zero execution time for each plan skeleton, that is, $E_k^1 = 0$ with probability 1, and set a deadline $D = W$.² Finally, the planning time CDF P_k^1 is defined as a piecewise constant function:

$$P_k^1(t) = \begin{cases} 0, & t < w_k, \\ \epsilon v_k, & w_k \leq t \leq W, \\ 1, & W < t. \end{cases}$$

Here, plan skeleton k has a probability of 0 of completing elaboration and execution before time w_k , a probability of ϵv_k of completing elaboration and execution after time w_k , and the probability increases further to 1 after the deadline. Later in the proof, we set a particular value

²The proof can be easily adapted to support any constant execution time.

for ϵ such that the probability of success of at least one plan skeleton is *almost* as much as the sum of the success probabilities of all the selected skeletons.

To prove NP-hardness, we establish bounds on the probability of success, denoted $P_{SUCC}(\mathcal{S}^\circ)$, for any set of items \mathcal{S}° :

$$\begin{aligned} \epsilon \sum_{s_m \in \mathcal{S}^\circ} v_m &\geq P_{SUCC}(\mathcal{S}^\circ), \\ &= 1 - \prod_{s_m \in \mathcal{S}^\circ} (1 - \epsilon v_m), \\ &> \epsilon \left(\sum_{s_m \in \mathcal{S}^\circ} v_m - 1 \right). \end{aligned}$$

The first inequality follows from the union bound, stating that the sum of event probabilities, where the probability of each event for plan skeleton m corresponds to ϵv_m , is always at least as large as the probability of the union of these events, implying that at least one plan skeleton finishes elaboration and execution.

The last inequality is demonstrated by bounding the higher-order terms when expanding the product into a sum. Now, let's proceed with the detailed proof for this inequality. First, we can derive the following equality using the principle of inclusion-exclusion:

$$\begin{aligned} P_{SUCC}(\mathcal{S}^\circ) &= 1 - \prod_{m=1}^M (1 - \epsilon v_m), \\ &= \sum_{m=1}^M \epsilon^m (-1)^{m+1} \sum_{\substack{\mathcal{N} \subseteq \{1, \dots, M\}, \\ |\mathcal{N}|=m}} \prod_{n \in \mathcal{N}} v_n. \end{aligned}$$

All the summands in the outer summation with odd m are positive, so if we exclude them (for all $m > 1$), the value of the expression does not increase.

$$\begin{aligned}
P_{SUCC}(\mathcal{S}^\circ) &= \sum_{m=1}^M \epsilon^m (-1)^{m+1} \sum_{\substack{\mathcal{N} \subseteq \{1, \dots, M\}, n \in \mathcal{N} \\ |\mathcal{N}|=m}} \prod v_n, \\
&\geq \epsilon \sum_{\substack{\mathcal{N} \subseteq \{1, \dots, M\}, n \in \mathcal{N} \\ |\mathcal{N}|=1}} \prod v_n - \sum_{m'=1}^{\lfloor \frac{M}{2} \rfloor} \epsilon^{2m'} \sum_{\substack{\mathcal{N} \subseteq \{1, \dots, M\}, n \in \mathcal{N} \\ |\mathcal{N}|=2m'}} \prod v_n, \\
&= \epsilon \sum_{m=1}^M v_m - \sum_{m'=1}^{\lfloor \frac{M}{2} \rfloor} \epsilon^{2m'} \sum_{\substack{\mathcal{N} \subseteq \{1, \dots, M\}, n \in \mathcal{N} \\ |\mathcal{N}|=2m'}} \prod v_n.
\end{aligned}$$

On the other hand, the number of elements in $\sum_{\substack{\mathcal{N} \subseteq \{1, \dots, M\}, \\ |\mathcal{N}|=2m'}} \prod_{n \in \mathcal{N}} v_n$ is $\binom{M}{2m'}$, which is upper-bounded by $M^{2m'}$. Let $H = \max_{m=1}^M v_m$. Since $H \geq v_m$ for all $1 \leq m \leq M$, we have $H^{2m'} \geq \prod_{n \in \mathcal{N}} v_n$. We obtain the following inequalities using these relations:

$$\begin{aligned}
P_{SUCC}(\mathcal{S}^\circ) &\geq \epsilon \sum_{m=1}^M v_m - \sum_{m'=1}^{\lfloor \frac{M}{2} \rfloor} \epsilon^{2m'} \sum_{\substack{\mathcal{N} \subseteq \{1, \dots, M\}, n \in \mathcal{N} \\ |\mathcal{N}|=2m'}} \prod v_n, \\
&\geq \epsilon \sum_{m=1}^M v_m - \sum_{m'=1}^{\lfloor \frac{M}{2} \rfloor} \epsilon^{2m'} \sum_{\substack{\mathcal{N} \subseteq \{1, \dots, M\}, \\ |\mathcal{N}|=2m'}} H^{2m'}, \\
&\geq \epsilon \sum_{m=1}^M v_m - \sum_{m'=1}^{\lfloor \frac{M}{2} \rfloor} (\epsilon H M)^{2m'}.
\end{aligned}$$

By setting $\epsilon = \frac{1}{H^2 M^3}$, we obtain:

$$\begin{aligned}
P_{SUCC}(\mathcal{S}^\circ) &\geq \epsilon \sum_{m=1}^M v_m - \sum_{m'=1}^{\lfloor \frac{M}{2} \rfloor} (\epsilon H M)^{2m'}, \\
&= \epsilon \sum_{m=1}^M v_m - \epsilon \sum_{m'=1}^{\lfloor \frac{M}{2} \rfloor} \epsilon^{2m'-1} (H M)^{2m'}, \\
&\geq \epsilon \left(\sum_{m=1}^M v_m - \sum_{m'=1}^{\lfloor \frac{M}{2} \rfloor} \frac{(H M)^{2m'}}{(H^2 M^3)^{2m'-1}} \right), \\
&= \epsilon \left(\sum_{m=1}^M v_m - \sum_{m'=1}^{\lfloor \frac{M}{2} \rfloor} \frac{1}{H^{2m'-2} M^{4m'-3}} \right), \\
&\geq \epsilon \left(\sum_{m=1}^M v_m - \sum_{m'=1}^{\lfloor \frac{M}{2} \rfloor} \frac{1}{M} \right), \\
&> \epsilon \left(\left(\sum_{s_m \in \mathcal{S}^\circ} v_m \right) - 1 \right),
\end{aligned}$$

where the second-to-last inequality can be derived from the fact that $\frac{1}{H^{2m'-2} M^{4m'-3}} \leq \frac{1}{M}$, since $H \geq v_k \geq 1$, $M \geq 1$, and $m' \geq 1$.

Next, we demonstrate that optimal schedules correspond to optimal knapsack solutions. Let \mathcal{S}° be an optimal schedule. We first show that \mathcal{S}° corresponds to a knapsack solution. Since the deadline of all plan skeletons is W , the processing time assigned to each plan skeleton $s_m \in \mathcal{S}^\circ$ cannot exceed W . Therefore, any plan skeleton assigned non-zero processing time in \mathcal{S}° receives time equal to w_m . Additionally, the sum of processing times in \mathcal{S}° is at most W . We use \mathcal{S}° to also denote the set of items in the knapsack problem corresponding to the plan skeletons, each assigned time w_m in \mathcal{S}° . The knapsack value of \mathcal{S}° is $V = \sum_{s_m \in \mathcal{S}^\circ} v_m$, and since the sum of weights of the items in \mathcal{S}° is at most W , \mathcal{S}° is a knapsack solution.

The remaining proof is to show that \mathcal{S}° is an optimal knapsack solution. Assume, in contradiction, that \mathcal{S}° is suboptimal as a knapsack solution. Then there must exist a knapsack solution \mathcal{S}^+ with value $V^+ \geq V + 1$. Since \mathcal{S}^+ is a knapsack solution, $\sum_{s_m \in \mathcal{S}^+} w_m \leq W$. Thus, taking \mathcal{S}^+ as a schedule (assigning time w_m to each plan skeleton $s_m \in \mathcal{S}^+$) creates a schedule where

all plan skeletons run before time W as well, with success probability:

$$\begin{aligned} P_{SUCC}(\mathcal{S}^+) &= 1 - \prod_{s_m \in \mathcal{S}^+} (1 - \epsilon v_m) > \epsilon \left(\sum_{s_m \in \mathcal{S}^+} v_m - 1 \right), \\ &\geq \epsilon \left(\sum_{s_m \in \mathcal{S}^\circ} v_i \right) \geq P_{SUCC}(\mathcal{S}^\circ), \end{aligned}$$

where the inequalities are due to the previously established bounds. That is, schedule \mathcal{S}^+ has a greater success probability than \mathcal{S}° , a contradiction. \square

V. MODEL-BASED APPROACH

Solving the proposed MDP-based effort allocation formulation requires access to planning time distributions and execution time distributions of all abstract actions involved in plan skeletons. Since these distributions are unknown a priori, we must either learn them from past experiences or data, or find a way to bypass explicit learning for finding a policy.

To address this challenge, we explore both *model-based* and *model-free* approaches. In model-based learning, we collect data on both planning times and execution times of all abstract actions taken in the training problems and estimate their distributions. In model-free learning, we utilize policy optimization from reinforcement learning to bypass distribution learning. We will delve into the model-free approach in the next section.

Both planning time distribution P_k^j and execution time distribution E_k^j are discrete, each modeled as a *categorical* or *multinoulli* distribution with parameter vector $\theta_{k,j}^{P/E}$ in the D -simplex, as each distribution contains $D + 1$ categories as a support. D parameters are positive real that sum to 1. The value of $D + 1$ implies non-plannable or non-executable within a deadline, relaxing the downward refinability assumption explained in Section III-C.

We compute maximum likelihood estimation of $\theta_{k,j}^{P/E}$ parameters for all k and j by counting the number of the corresponding events in the data. Laplace smoothing can also be applied to make robust estimation if data is sparse.

Using the estimated distributions, we are now prepared to solve the proposed MDP problem. This problem-solving incurs meta-level computation, where smaller is better as the planner can allocate its resources more efficiently to object-level computations, corresponding to planning and execution times. However, as demonstrated in Section IV, optimally solving the proposed MDP using, for example, value iteration, is computationally intractable. Therefore, we propose two approximate schemes for efficiently solving the MDP.

The first approach is approximating value functions via MCTS. The second approach is to compute the optimal *linear contiguous* policy, which is a linear policy in which all time allocations to the same plan skeleton are performed contiguously.

A. Monte Carlo Tree Search

We employ MCTS [75, 76] to handle the exponential increase of states in the proposed MDP, as shown in Theorem 1. To overcome the curse of dimensionality, MCTS leverages random episode sampling within a decision tree to approximate an optimal policy that would be found by an expensive value iteration algorithm.

A decision tree of MCTS when applied to solving MDP problems corresponds to an *expectimax* tree, where the tree nodes, starting from the root, are structured as an alternating sequence of action-choice nodes and probabilistic nodes. The value of a probabilistic node is computed as the expectation of the values of its children nodes, and the value of an action-choice node is computed as the maximum of the values of its children. The unique feature of the expectimax tree constructed for the proposed MDP is that every action-choice node always has K children nodes, as the action is about determining which plan skeleton to refine among K plan skeletons regardless of the state being considered, except for terminal states.

Value iteration utilizes the optimal state-action value function $Q^* : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, defined as:

$$Q^*(s, a) = R(s, a) + \sum_{s' \in \mathcal{S}} T(s, a, s') V^*(s'),$$

where V^* denotes the optimal state value function. An optimal policy can then be defined as $\pi(s) = \arg \max_{a \in \mathcal{A}} Q(s, a)$.

On the other hand, MCTS approximates the state-action value function Q using random simulation. In particular, we adopt the upper confidence bounds applied for trees (UCT [77]) as an MCTS algorithm, which treats action choice as a *K-armed bandit* problem for choosing from K plan skeletons and aims to achieve the property that the probability of selecting a sub-optimal action converges to 0 as the computation time goes to infinity by balancing exploration-exploitation trade-off. This trade-off is captured in the form of:

$$\pi(s) = \arg \max_{a \in \mathcal{A}} \left\{ Q(s, a) + C \sqrt{\frac{\ln N(s)}{N(s, a)}} \right\},$$

where $N(s)$ denotes the number of times state s has been visited, and $N(s, a)$ denotes the number of times action a has been sampled in state s , both in previous iterations. $C > 0$ denotes an exploration constant, where an increased value implies encouraging exploration, while a decreased value implies encouraging exploitation.

We apply a general MCTS framework iteratively running four phases: (1) *selection*, where a node in the tree that is not yet explored is selected based on the above UCT criteria, (2) *expansion*, where a chosen node is expanded by applying an available action, (3) *simulation*, where random actions are applied to the node expanded from the chosen node until it reaches a terminal state, and (4) *backpropagation*, where the reward accumulated in the episode generated by the simulation phase is backpropagated up to the root node. The accumulated reward corresponds to the number of successful rollouts that meet a pre-specified deadline. The policy quality can primarily be determined by the allotted MCTS computation time and the number of rollouts used in the simulation phase.

MCTS is an *anytime* algorithm, implying that the more computation time is exploited, the closer the computed policy is to the optimal, and that it can be terminated at any time but still outputs an answer. This property is desirable as users can determine the meta-level computation time for their applications at the sacrifice of policy quality. It is worth noting that finding a sweet spot between MCTS computation time and policy quality can be considered as meta-meta-reasoning, but we do not delve into this aspect in this work.

B. Heuristics

We introduce two properties that a policy may exhibit as heuristics, which systematically restrict the solution policy space. This restriction may result in identifying a different optimal policy than the one found by solving the proposed MDP, thereby forming an approximate solution, but it significantly facilitates the design of an efficient algorithm. The first property is *linear*, and the second property is *contiguous*.

A linear policy is a pre-determined allocation of time steps to all plan skeletons, which remains unchanged regardless of the refinement process unless a valid solution is found. The optimal policy in Figure 1 (2) is not linear, as the decision of which action to allocate time to depends on the result of the previous allocation (*e.g.*, we cannot know in advance whether action δ_1^2 or δ_3^1 will be allocated computation time after δ_1^1 or δ_2^1).

A contiguous policy is one in which all time allocations to the refinement of an action are performed contiguously, without any other allocations in between. The optimal policy in Figure 1 (2) is contiguous, whereas the linear policy $(\delta_3^1, \delta_1^1, \delta_3^1)$, for example, is not contiguous.

Consider the special case where each plan skeleton consists of only a single action and the execution times are known. In this case, an action δ_k^j should never be executed if its execution time plus the current time exceeds the deadline, as it will never lead to a timely solution. Consequently, there exists an optimal solution that excludes such actions. When considering policies that only schedule computation actions if they can lead to a timely solution, the result of each computation allocation to an action δ_k^j is either: (1) a successful elaboration of action δ_k^j , after which plan skeleton k is fully elaborated and can be executed before the deadline, allowing the algorithm to terminate; or (2) action δ_k^j is not elaborated. Since only option (2) results in additional time allocations, the resulting policy is linear. Moreover, linear non-contiguous policies can be rearranged as contiguous without affecting the probability of success. Thus, in this special case, there exists an optimal policy that is both linear and contiguous.

While linear contiguous policies are not necessarily optimal in the general case, they can be computed in pseudo-polynomial time [78]. Consequently, we can compute a linear contiguous policy and use it as a (potentially suboptimal) solution to the effort allocation problem to efficiently find a solution.

To compute this policy, we introduce the dynamic programming (DP) scheme defined in Equation 1. $N(k, l_k + 1)$ denotes the index set of plan skeletons that share the action $\delta_k^{l_k+1}$. For example, in Figure 1 (1), $N(1, 1) = \{1, 2\}$ since $\delta_1^1 = \delta_2^1$, and $N(3, 1) = \{3\}$. PS represents a DP state, indicating the probability of successful refinement associated with that state.

Specifically, $PS(k, 0, 0, 0)$ represents the probability that plan skeleton k will be completed on time under the linear contiguous policy. Equation 1a serves as the termination condition, where the last action of the plan skeleton $\delta_k^{A_k}$ is being considered for refinement. All possible time allocations up to the deadline for $\delta_k^{A_k}$ are evaluated, with each allocation using the remaining time steps before the deadline as execution time, as no further refinement will be needed if it succeeds. Equation 1b forms the core of the DP scheme. In this case, we again consider all possible time allocations up to the deadline for the current refining action $\delta_k^{l_k+1}$ (first summation). For each time allocation, we evaluate all possible execution times for $\delta_k^{l_k+1}$ (second summation). For each combination of time allocation and execution time, we apply the maximum value of

$$PS(k, l_k, CT, ET_k) = \begin{cases} \sum_{t=1}^{D-CT} p_k^{l_k+1}(t) \cdot E_k^{l_k+1}(D - t - CT - ET_k) & \text{if } l_k + 1 = A_k, \quad (1a) \\ \sum_{t=1}^{D-CT} \sum_{m \in \text{Support}(e_k^{l_k+1})} \left(p_k^{l_k+1}(t) \cdot e_k^{l_k+1}(m) \right. \\ \cdot \max_{k' \in N(k, l_k+1)} PS(k', l_{k'} + 1, CT + t, ET_{k'} + m) \left. \right) & \text{if } l_k + 1 < A_k. \quad (1b) \end{cases}$$

the DP states that consider plan skeletons sharing the current action $\delta_k^{l_k+1}$, while adding the corresponding current and execution time steps.

We compute the values of $PS(k, 0, 0, 0)$ for all plan skeletons and select the one with the highest value, indicating the most probable successful refinement. Once the skeleton is chosen by the DP, it is applied for actual refinement within the available time steps, ensuring the deadline is met.

To mitigate the linearity assumption of DP, we introduce a variant of the algorithm called DP_Rerun, with the pseudocode provided in Algorithm 1. In this variant, at each decision-making time step, we first identify a plan skeleton that maximizes $PS(k, l_k, CT, ET_k)$ (lines 2-6), as done in DP. However, unlike DP, where the entire available time is allocated to the chosen skeleton, DP_Rerun allocates only a single computational action for refinement (line 7). We then observe the result: either the elaboration of the corresponding action is completed, providing the execution time (lines 12, 13), or the elaboration process remains incomplete (line 15). In the latter case, we update the planning time distribution of the action, accounting for the time spent without completing the refinement. This process of running DP, selecting actions, obtaining observations, and updating the distribution continues iteratively until either the problem is solved (line 10) or the deadline has passed (line 19). Although DP_Rerun requires more computation than DP due to its online replanning behavior, it mitigates the linear policy assumption and yields better solutions.

VI. MODEL-FREE APPROACH

In the model-based approach introduced in the previous section, we estimate the parameters of planning time and execution time distributions using maximum likelihood estimation to learn

Algorithm 1: DP_Rerun

Input : $CT = 0, l_1 = 0, \dots, l_K = 0, ET_1 = 0, \dots, ET_K = 0$ **Output**: SUCCESS or FAILURE

```

1 while  $CT + \min\{ET_1, \dots, ET_K\} < D$  do
2    $\Sigma.val \leftarrow \emptyset$ 
3   for  $0 \leq k \leq K$  do
4      $\Sigma.val.append(\text{solveDP}(PS(k, l_k, CT, ET_k)))$  // Apply Equations 1a
       and 1b.
5   end
6    $k^* \leftarrow \arg \max_k \Sigma.val$ 
7   refine( $\sigma_{k^*}$ ) // Allocate one time step to refine the  $k^*$ -th
       plan skeleton.
8   if isActionRefined( $l_{k^*} + 1$ ) then
9     if  $l_{k^*} = A_{k^*}$  then
10      return SUCCESS
11    end
12     $l_{k^*}++$ 
13     $ET_{k^*} \leftarrow ET_{k^*} + m$  // Here,  $m$  denotes the actual execution
       time derived from the refined action.
14  else
15    updateDistribution( $l_{k^*} + 1$ ) // Update the planning time
       distribution for the action  $\delta_{k^*}^{l_{k^*}+1}$ .
16  end
17   $CT++$ 
18 end
19 return FAILURE

```

the transition models T in the MDP. This approach requires gathering data offline to fit the parameters of the distributions to match the data. The learned transition models are then used to compute a plan-skeleton selection policy for a query problem.

However, we often face situations where data is not readily available before the deployment of the robot, yet the robot still needs to learn a selection policy for a query problem without access to distribution information and improve its policy while repeatedly solving the problem. This approach essentially describes *episodic reinforcement learning*. In this context, the agent operates on the same environment but is unaware of the underlying model (MDP). In this context, the agent interacts with the same environment but is unaware of the underlying model (MDP). The only information available to the agent is the state variables and the action space. The agent learns by observing the current state of the environment, performing an action, and observing the resulting next state and immediate reward.

In particular, we employ Proximal Policy Optimization (PPO [79]), a prominent variant within the family of policy gradient methods, due to its desirable properties such as sample efficiency, stable training, and ease of use. PPO improves on traditional policy gradient methods by using a surrogate objective function that penalizes large policy updates, ensuring more stable and reliable learning. This is achieved through techniques such as clipping the probability ratio between the new and old policies, which prevents drastic changes that could destabilize training. Additionally, PPO uses mini-batch updates and adaptive step sizes, contributing to its effectiveness and efficiency in various reinforcement learning tasks.

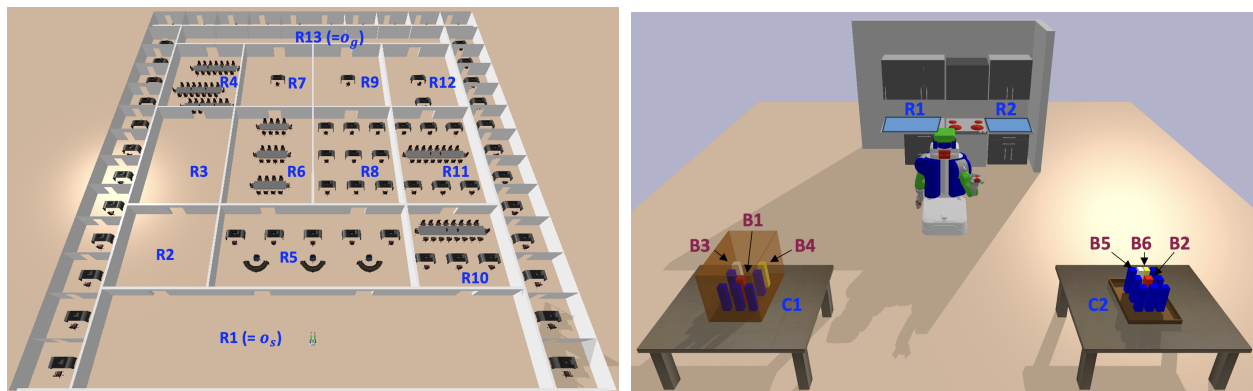
In policy gradient methods, a policy is parameterized by a neural network that takes an MDP state as input and outputs an MDP action or a distribution of actions. In our context, the policy determines which plan skeleton to refine for one time step. The policy network is trained by deploying the agent in the environment for a finite collection of episodes. During these episodes, the agent collects sequences of states, actions, and rewards (trajectories), and the model is trained iteratively using this data.

It is worth mentioning, however, that the proposed reinforcement learning problem is intractable due to *sparse* rewards. A reward of 1 is received only if the refinement of any plan skeleton is completed within a deadline, and a reward of 0 is received if it fails, as defined in the MDP described in Section III-C. Sparse reward problems, especially when the reward is only obtained at the end of the episode, are notoriously challenging. Without a reward signal until the episode’s conclusion, it is difficult for the policy to learn which actions were beneficial, complicating the process of assigning credit and making effective decisions. Moreover, due to the stringent time deadlines inherent in these problems, exploration-promoting mechanisms like

the Intrinsic Curiosity Module (ICM [80]) and Go-Explore [81], which are typically effective in handling sparse rewards, are unlikely to be viable here. Each action taken for exploration decreases the probability of meeting the deadline, posing a significant challenge in balancing exploration against the imperative of timely completion.

VII. EXAMPLE SCENARIOS

In this section, we provide two example scenarios implemented in PyBullet simulation [82] to demonstrate practical use cases of the proposed method and the specifications of the corresponding TAMP formulations. The first scenario is the navigation domain, where the goal is to reach a target office through a series of offices with different geometries. The second scenario is the manipulation domain, where the robot needs to move several objects to the kitchen area for meal preparation. In both scenarios, the planner must adhere to a pre-specified deadline. We will present experimental results using these domains in Section VIII-D.



(1) Navigation domain.

(2) Manipulation domain.

Fig. 2: Visualization of domains used in the example scenarios.

A. Navigation domain

The visualization of the navigation domain is included in Figure 2 (1), where the target office, start office, and intermediate offices are depicted. The navigation domain considers only fixed objects corresponding to individual office rooms, excluding movable objects, thus $\mathcal{O} = \mathcal{O}_F$.

We only utilize the mobile base of the PR2 to use it as a mobile robot. Therefore, the configuration space becomes $\mathbb{Q} \in SE(2)$, specifying (x, y, θ) . The planner has access to the

xy domain of the workspace of each office room and can determine if the robot is located in the office.

In this domain, four forms of predicates \mathcal{P} are used. Their corresponding literals are as follows:

- $\text{InRoom}(o_f \in \mathcal{O}_F, q \in \mathbb{Q})$, true if the robot, at configuration q , is within the office room o_f .
- $\text{Reachable}(q \in \mathbb{Q}, q' \in \mathbb{Q})$, true if a collision-free path between configurations q and q' is found.
- $\text{AtConf}(q \in \mathbb{Q})$, true if the robot is at configuration q .
- $\text{CFree}(q \in \mathbb{Q})$, true if the robot at configuration q is collision-free with any fixed objects.

InRoom , Reachable , and CFree are constant predicates, while AtConf is a fluent predicate as its state varies depending on configuration q .

Actions Δ include a single form, which is $\text{move}(o_f \in \mathcal{O}_F, o'_f \in \mathcal{O}_F, q \in \mathbb{Q}, q' \in \mathbb{Q})$. The preconditions and effects of move are:

- **pre:** $\text{InRoom}(o_f, q)$, $\text{Reachable}(q, q')$, $\text{AtConf}(q)$, $\text{CFree}(q)$.
- **eff:** $\text{InRoom}(o'_f, q')$, $\neg \text{AtConf}(q)$, $\text{AtConf}(q')$, $\text{CFree}(q')$.

Here, \neg denotes a negation symbol.

Application of move implies that the robot transitions from configuration q in office room o_f to configuration q' in office room o'_f , and a collision-free path between configurations q and q' is obtained as a byproduct of evaluating Reachable . q' is determined by uniform sampling within the 2D region of office room o'_f . We employ RRT-Connect [83] to compute a collision-free path.

Initial literals \mathcal{I} are $\{\text{InRoom}(o_s, q_{init}), \text{AtConf}(q_{init}), \text{CFree}(q_{init})\}$, where o_s denotes the start office room and q_{init} is the initial robot configuration. Goal literals \mathcal{G} are $\{\text{InRoom}(o_g, q \in \mathbb{Q}), \text{CFree}(q \in \mathbb{Q})\}$ for any configuration q where InRoom is true, and o_g denotes the goal office room.

There are four plan skeletons $\Sigma = \{\sigma_1, \dots, \sigma_4\}$ as follows:

- $\sigma_1 = (\delta_1^1 = \text{move}(o_f = R_1, o'_f = R_2), \delta_1^2 = \text{move}(o_f = R_2, o'_f = R_3), \delta_1^3 = \text{move}(o_f = R_3, o'_f = R_4), \delta_1^4 = \text{move}(o_f = R_4, o'_f = R_{13}))$,
- $\sigma_2 = (\delta_2^1 = \text{move}(o_f = R_1, o'_f = R_5), \delta_2^2 = \text{move}(o_f = R_5, o'_f = R_6), \delta_2^3 = \text{move}(o_f = R_6, o'_f = R_7), \delta_2^4 = \text{move}(o_f = R_7, o'_f = R_{13}))$,
- $\sigma_3 = (\delta_3^1 = \text{move}(o_f = R_1, o'_f = R_5), \delta_3^2 = \text{move}(o_f = R_5, o'_f = R_8), \delta_3^3 = \text{move}(o_f =$

- $$R_8, o'_f = R_9), \delta_3^4 = \text{move}(o_f = R_9, o'_f = R_{13})),$$
- $\sigma_4 = (\delta_4^1 = \text{move}(o_f = R_1, o'_f = R_{10}), \delta_4^2 = \text{move}(o_f = R_{10}, o'_f = R_{11}), \delta_4^3 = \text{move}(o_f = R_{11}, o'_f = R_{12}), \delta_4^4 = \text{move}(o_f = R_{12}, o'_f = R_{13})).$

where arguments for typed variables are omitted from actions. Notice that σ_2 and σ_3 plan skeletons share the same action $\text{move}(o_f = R_1, o'_f = R_5)$.

B. Manipulation domain

Unlike the navigation domain, the manipulation domain includes movable objects \mathcal{O}_M , as well as three fixed objects $\mathcal{O}_F = \{\text{Worktop}, \text{Table}, \text{Shelf}\}$, as visualized in Figure 2 (2). Here, the goal is to move two movable objects, one from the shelf (*i.e.*, B1) and another from the table (*i.e.*, B2), to the worktop. However, other movable objects are located near B1 and B2, potentially resulting in a long planning time for directly retrieving B1 and B2, which may not meet a deadline. Instead, removing nearby movable objects first may be beneficial, as it creates space and shortens the planning time for grasping B1 and B2.

In this domain, the robot's configuration space is $\mathcal{Q} \in SE(2) \times \mathbb{T}^7$, where $SE(2)$ represents the configuration for the mobile base of the PR2, as used in the navigation domain, and \mathbb{T}^7 represents the configuration of the 7 joints of the PR2's left arm (*i.e.*, 7-dimensional torus). The pose space of each movable object is $\mathcal{P}_i \in SE(3)$, specifying rigid body transformations in 3D space.

More predicates \mathcal{P} and actions Δ are introduced to describe physical interactions between the robot and the objects. In addition to `AtConf` introduced in the navigation domain, the literals of the additional predicates are as follows:

- $\text{InRegion}(o_i \in \mathcal{O}_M, o_f \in \mathcal{O}_F, p_i \in \mathbb{P}_i)$, true if the movable object o_i with pose p_i is in the workspace of the fixed object o_f .
- $\text{Reachable}(q \in \mathbb{Q}, q' \in \mathbb{Q}, \forall i p_i \in \mathbb{P}_i)$, true if a collision-free path between configurations q and q' is found with all movable objects at poses p_i .
- $\text{Grasp}(o_i \in \mathcal{O}_M, p_i \in \mathbb{P}_i, g \in \mathbb{G})$, true if the movable object o_i at pose p_i can be grasped by the robot at grasp pose g .
- $\text{Kin}(o_i \in \mathcal{O}_M, p_i \in \mathbb{P}_i, g \in \mathbb{G}, q \in \mathbb{Q})$, true if the robot at configuration q , with grasp pose g , and the movable object o_i at pose p_i , satisfies a kinematic constraint.

- $\text{InHand}(o_i \in \mathcal{O}_M, g \in \mathbb{G})$, true if the movable object o_i is stably grasped by the robot with grasp pose g .
- Empty , true if the robot's hand is empty.
- $\text{AtPose}(o_i \in \mathcal{O}_M, p_i \in \mathbb{P}_i)$, true if the movable object o_i is at pose p_i .
- $\text{CFree}(q \in \mathbb{Q}, \forall i p_i \in \mathbb{P}_i)$, true if the robot at configuration q is collision-free with all fixed objects and movable objects at poses p_i .

InRegion , Reachable , Grasp , Kin , and CFree are constant predicates, while InHand , Empty , AtPose , and AtConf are fluent predicates.

The following three forms of actions are used in the manipulation domain:

- $\text{move}(q \in \mathbb{Q}, q' \in \mathbb{Q}, \forall i p_i \in \mathbb{P}_i)$
 - **pre**: $\text{Reachable}(q, q', \forall i p_i)$, $\text{AtConf}(q)$, $\text{CFree}(q, \forall i p_i)$.
 - **eff**: $\neg \text{AtConf}(q)$, $\text{AtConf}(q')$, $\text{CFree}(q', \forall i p_i)$.
- $\text{pick}(o_i \in \mathcal{O}_M, p_i \in \mathbb{P}_i, g \in \mathbb{G}, q \in \mathbb{Q})$
 - **pre**: $\text{AtPose}(o_i, p_i)$, $\text{Kin}(o_i, p_i, g, q)$, Empty , $\text{Grasp}(o_i, p_i, g)$, $\text{AtConf}(q)$.
 - **eff**: $\neg \text{AtPose}(o_i, p_i)$, $\neg \text{Empty}$, $\text{InHand}(o_i, g)$.
- $\text{place}(o_i \in \mathcal{O}_M, o_f \in \mathcal{O}_F, p_i \in \mathbb{P}_i, g \in \mathbb{G}, q \in \mathbb{Q})$
 - **pre**: $\text{Kin}(o_i, p_i, g, q)$, $\text{InHand}(o_i, g)$, $\text{AtConf}(q)$.
 - **eff**: $\text{InRegion}(o_i, o_f, p_i)$, $\text{AtPose}(o_i, p_i)$, Empty , $\neg \text{InHand}(o_i, g)$.

The move action is similar to the same action in the navigation domain, except that CFree additionally evaluates collisions with objects. RRT-Connect is used to compute a collision-free path. pick and place actions involve evaluating Kin to find inverse kinematic solutions, for which we use IKFast [64]. The pick action evaluates Grasp , where a grasp is sampled with respect to the tool frame of the PR2. The place action requires sampling a placement pose p_i of the movable object o_i in the workspace space of the fixed object o_f , such as the surface of the table.

Initial literals \mathcal{I} are $\{\text{AtConf}(q_{init}), \text{Empty}, \text{CFree}(q_{init}, \forall i p_{i,init}), \text{InRegion}(B1, C1, p_{1,init}), \text{InRegion}(B2, C2, p_{2,init}), \text{InRegion}(B3, C1, p_{3,init}), \text{InRegion}(B4, C1, p_{4,init}), \text{InRegion}(B5, C2, p_{5,init}), \text{InRegion}(B6, C2, p_{6,init}), \text{AtPose}(o_i \in \mathcal{O}_M, p_{i,init} \in \mathbb{P}_i)\}$. Goal literals \mathcal{G} are $\{\text{InRegion}(B1, R1, p_1 \in \mathbb{P}_1), \text{InRegion}(B2, R2, p_2 \in \mathbb{P}_2)\}$.

Eight plan skeletons, $\Sigma = \{\sigma_1, \dots, \sigma_8\}$, are given as input to this domain. We include the

details of the plan skeletons in Appendix A due to limited space.

VIII. EXPERIMENTS

In the experiments, we primarily evaluate two key questions: (1) How effectively can our method find a deadline-aware executable plan compared to baseline methods? (2) How efficiently can our heuristic schemes determine the computation allocation policy compared to approaches that attempt to solve the proposed MDP, which has been proven to be NP-hard? Both questions assess our method’s performance as an effort allocation scheduler and its computational efficiency.

To address these questions, we design several problem instances for case studies to systematically evaluate the performance of the proposed methods, measured by the statistics of rewards collected from episodes. We also propose several baselines for effective performance comparison. The comparative analysis is conducted with all methods aiming to maximize collected rewards given sufficient computation time. This approach allows us to focus on evaluating the maximum capacity of each method without the constraint of limited computation time.

To complement the comparison analysis, we present an analysis of the computation time for the model-based approaches to validate the efficiency improvements introduced by the heuristics compared to MCTS. Lastly, evaluation results on the example scenarios introduced in Section VII are shown again under the sufficient computation time regime.

A machine with an Intel Core i7-12700H CPU @ 4.60GHz and 32 GB of memory was used for the experiments. For MCTS, we set the exploration constant C to 0.5. For PPO, we use a codebase from OpenAI Spinning Up [84] and include the details of the network architectures for the actor and critic, as well as the hyperparameters, in Appendix B.

A. Baselines

We design two additional algorithms as baselines to compare with the proposed algorithms. These baselines are simple and thus expected to perform as lower bounds. The first baseline is Round Robin, which deterministically chooses a plan skeleton for refinement at each time step from a pre-specified order among plan skeletons without needing to learn any distributions or repeat problem-solving. Specifically, Round Robin repeatedly selects from σ_1 to σ_K . The second baseline is Greedy, which, like model-based methods, needs to learn distributions and always

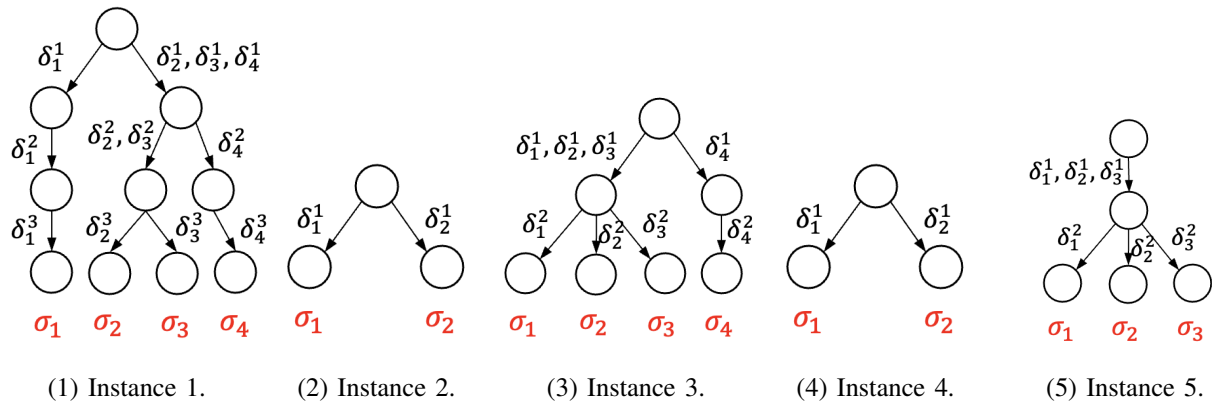


Fig. 3: Problem instances designed for the experiments.

chooses the plan skeleton with the smallest sum of the mean planning and execution times of all involved actions.

B. Comparison analysis

Figure 3 depicts five problem instances used for comparison analysis. Instance 1 contains several shared actions, but the mean of planning and execution times is equal for all plan skeletons. Instance 2 also has the same mean for both plan skeletons, and the distributions are symmetric but differ in the variance of planning time distributions. Instance 3 contains a shared action, but the plan skeletons do not have the same mean. Instance 4 is similar to Instance 2 except that the distributions are asymmetric. Instance 5 includes infeasible plan skeletons that cannot be refined even if all available time steps are dedicated. The deadlines pre-specified for these five problem instances are 14, 9, 20, 4, and 14, respectively. Histograms of planning time and execution time for all actions in the five problem instances are provided in Appendix C.

The performance is measured by the 95% confidence interval of the return (or cumulative reward) over 100 runs in each problem instance. We do not include the approach of solving the proposed MDP exactly, as its computation time is incomparably greater than all other methods, making it impractical in practice.

Table I shows the results of all methods across five problem instances, each given sufficient computation time to attempt their best performance. MCTS performed the best in all instances, reaching an optimal solution due to the sufficient computation time provided for each iteration of

Problem instance 1						
Algorithm	MCTS	DP	DP_Rerun	PPO	Round Robin	Greedy
Cumulative reward	0.98 ± 0.02	0.93 ± 0.05	0.94 ± 0.05	0.72 ± 0.09	0.37 ± 0.09	0.92 ± 0.05
Problem instance 2						
Algorithm	MCTS	DP	DP_Rerun	PPO	Round Robin	Greedy
Cumulative reward	0.74 ± 0.09	0.65 ± 0.09	0.70 ± 0.09	0.37 ± 0.09	0.51 ± 0.09	0.65 ± 0.09
Problem instance 3						
Algorithm	MCTS	DP	DP_Rerun	PPO	Round Robin	Greedy
Cumulative reward	0.71 ± 0.09	0.35 ± 0.09	0.67 ± 0.09	0.25 ± 0.08	0.59 ± 0.09	0.34 ± 0.09
Problem instance 4						
Algorithm	MCTS	DP	DP_Rerun	PPO	Round Robin	Greedy
Cumulative reward	0.94 ± 0.05	0.90 ± 0.06	0.90 ± 0.06	0.79 ± 0.08	0.45 ± 0.09	0.54 ± 0.09
Problem instance 5						
Algorithm	MCTS	DP	DP_Rerun	PPO	Round Robin	Greedy
Cumulative reward	0.71 ± 0.09	0.33 ± 0.09	0.70 ± 0.09	0.10 ± 0.06	0.70 ± 0.09	0.34 ± 0.09

TABLE I: Performance comparison among algorithms in five problem instances.

its four phases. DP_Rerun achieved the second-best performance with only marginal degradation compared to MCTS. This result is surprising, as we will show in the next subsection on the computation analysis between MCTS and DP_Rerun, where DP_Rerun required only negligible computation time. The performance of DP was comparable to DP_Rerun but worse in instances 3 and 5. This was caused by the potential inability to refine the chosen plan skeleton within a deadline, highlighting the importance of rerunning the computation at every time step. From these results, we observe that the contiguity heuristic works effectively in practice, while the linearity heuristic may face difficulty in some scenarios.

Greedy performed comparably to the model-based methods in instances 1 and 2 but poorly in the remaining instances. As this method only relies on the mean of distributions, problems like instances 3, 4, and 5 with heavy-tailed distributions that could lead to an unrefinable plan within a deadline can deceive Greedy, underscoring a major limitation of this method.

PPO and Round Robin performed equally poorly. PPO required 10^4 trials to train the model, which is quite extensive, to achieve this performance. Its computation time was insignificant, as it only required time for the forward pass of the neural network at each time step. While the

model-free variant has its value in handling data acquisition issues, dealing with sparse rewards in the presence of deadline constraints remains a significant challenge, left for future work.

Contrary to expectations, Round Robin performed well in instances 3 and 5, likely due to the small number of plan skeletons and deadlines used in the comparison analysis. However, in subsequent analyses involving larger plan skeletons and deadlines in example domains, Round Robin did not succeed even once.

C. Computation time analysis

As metareasoning trades off between solution quality and computation time, better solution quality can be achieved with more computation time. Since the model-free approach inherently differs from the model-based approach by requiring short computation time at the expense of a large number of problem-solving trials, we compare MCTS and DP_Rerun for this analysis. The more computation time used for MCTS, the closer its solution approaches the optimal solution that can be obtained by solving the MDP exactly.

Figure 4 shows the total computation times spent solving the five problem instances used in Section VIII-B. It can be observed that MCTS required greater computation time to find good-quality solutions, while DP_Rerun needed negligible computation time to find comparable solutions, demonstrating the dramatic increase in efficiency achieved by the heuristics.

In Figure 5, we compare the performance, measured by cumulative rewards over 100 runs, between MCTS and DP_Rerun. Both methods are given the same computation time, which is set equal to DP_Rerun’s computation time. This analysis evaluates the solution quality when both methods are constrained by fixed computation time. The significant degradation in MCTS’s performance can be observed, emphasizing DP_Rerun’s ability to maintain reasonable performance even with approximations.

We additionally evaluate the performance of MCTS by varying the computation time across 100 runs of five problem instances. Since each problem instance requires different computation times to find high-quality solutions, we use the logarithm of the computation time multiplier. This multiplier can be applied to compute the actual computation time in seconds for each environment by multiplying a base computation time for DP_Rerun with ten raised to the power of the multiplier value. The base computation times for DP_Rerun for each instance are 1.11×10^{-5} , 1.26×10^{-5} , 2.07×10^{-5} , 1.27×10^{-5} , 2.12×10^{-5} . More importantly, we can empirically

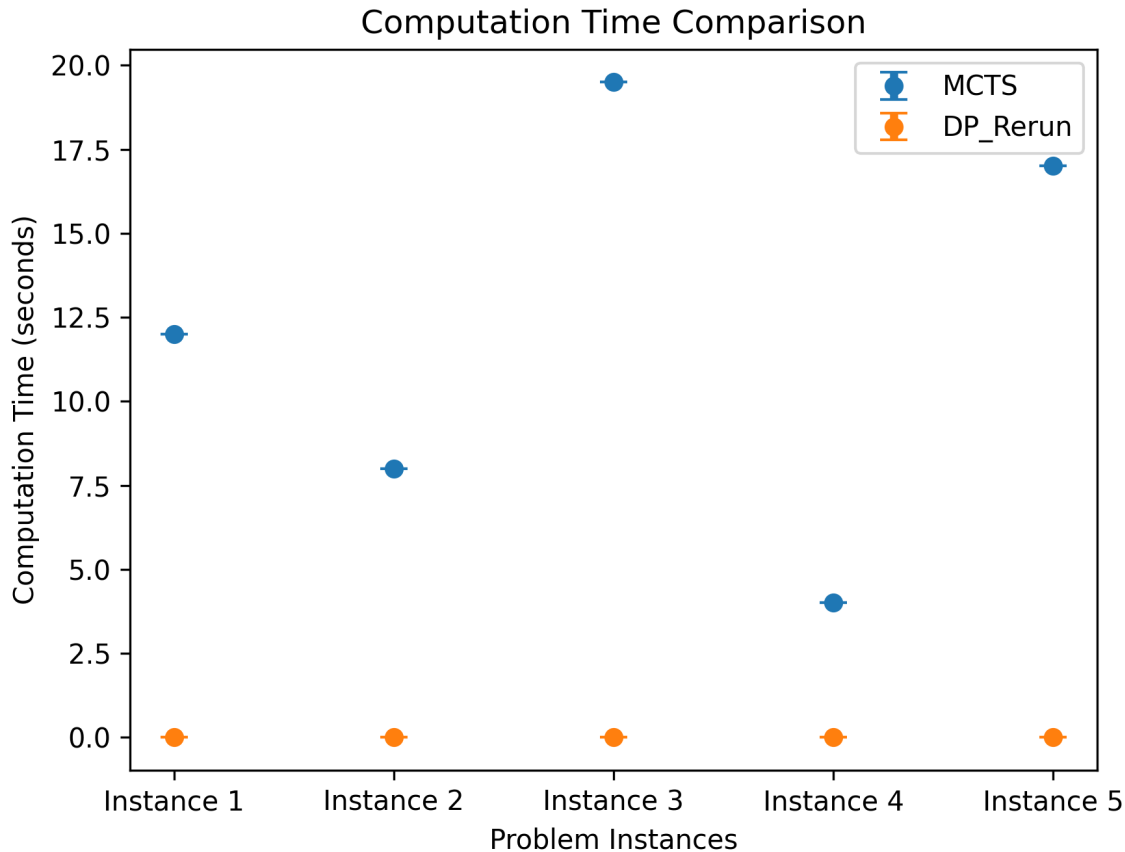


Fig. 4: Total computation time comparison between MCTS and DP_Rerun.

observe the fundamental trade-off in metareasoning, where more computation time leads to better solution quality, in this MCTS performance analysis.

D. Evaluation on example domains

We present the comparison analysis of all the methods in the example domains introduced in Section VII. Histograms of planning time and execution time for all actions in both the navigation and manipulation domains are provided in Appendix C. The performance is again measured by the 95% confidence interval of the return over 100 runs. The total decision-making time step D is set to 22 for navigation and 40 for manipulation. Like Section VIII-B, we provided sufficient computation time for all methods to perform at their best.

Table II shows the performance results of all methods. In both domains, DP_Rerun performed

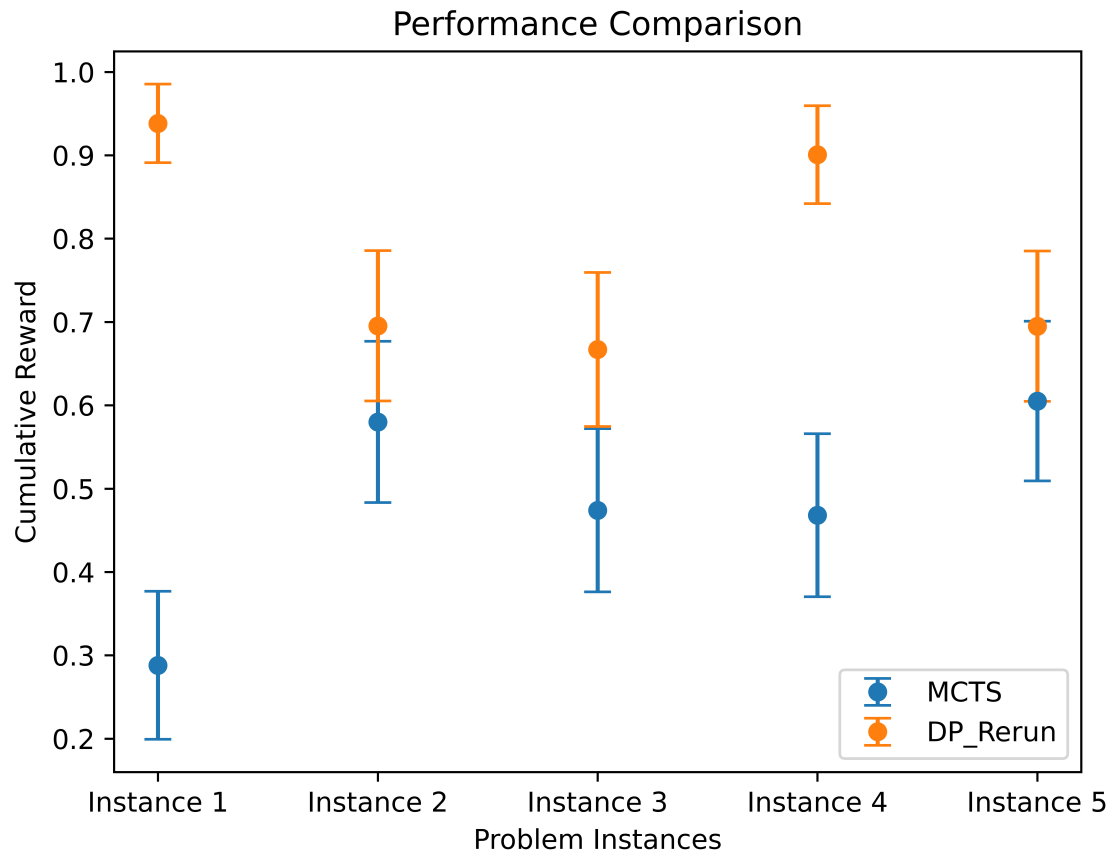


Fig. 5: Performance comparison between MCTS and DP_Rerun when both methods are given the same computation time.

the best, followed by DP and Greedy. The improved performances of DP and Greedy compared to those in the five instances are attributed to the tails of distributions not being heavy enough to reduce the probability of plans becoming unrefinable within a deadline, thereby increasing their success rates. Unlike the previous comparison analysis, MCTS performed poorly. This dramatic decrease in performance was due to insufficient computation time, even though it was given significantly more time compared to other methods, averaging 21 seconds for navigation and 45 seconds for manipulation. This result underscores the real complexity of the NP-hard effort allocation problem in large-scale scenarios. PPO was again trained with 10^4 trials, performing well in navigation but poorly in manipulation due to the large deadline and action space associated

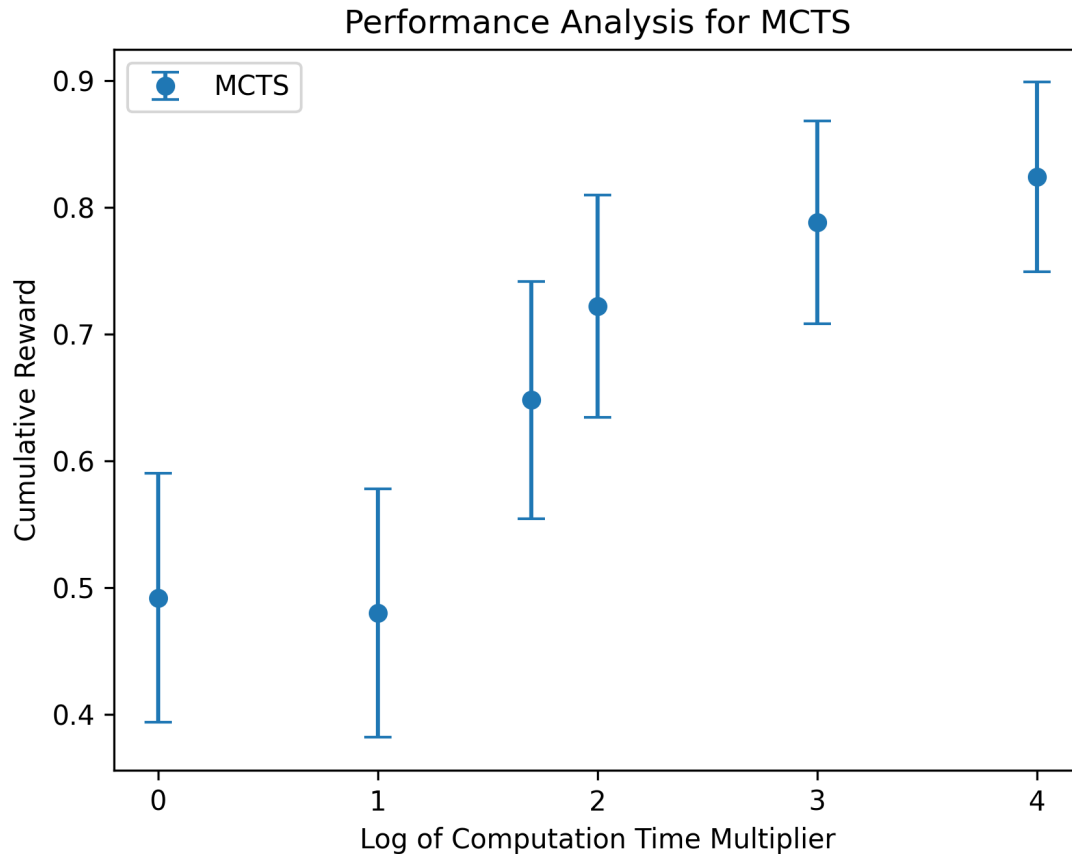


Fig. 6: Performance analysis of MCTS with varying computation time.

with a greater number of plan skeletons.

IX. CONCLUSION AND FUTURE WORK

This work addresses deadline-aware TAMP problems with the objective of finding a fully executable plan from a set of abstract plans without violating a pre-specified deadline constraint. A metareasoning approach is proposed by formulating an MDP to find optimal effort allocation among abstract plans. Since solving the MDP is proven to be NP-hard, we propose several approximation schemes and further explore a model-free approach that does not require learning the distributions. The proposed methods, particularly DP_Rerun, show promising results compared to baselines in terms of the cumulative reward related to solution quality and computation time.

Navigation domain						
Algorithm	MCTS	DP	DP_Rerun	PPO	Round Robin	Greedy
Cumulative reward	0.25 ± 0.08	0.43 ± 0.09	0.48 ± 0.09	0.42 ± 0.09	0.0 ± 0.0	0.43 ± 0.09
Manipulation domain						
Algorithm	MCTS	DP	DP_Rerun	PPO	Round Robin	Greedy
Cumulative reward	0.15 ± 0.07	0.47 ± 0.09	0.53 ± 0.09	0.10 ± 0.06	0.0 ± 0.0	0.48 ± 0.09

TABLE II: Performance comparison among algorithms in two example scenarios.

As the proposed effort allocation problem introduces a deadline constraint for the first time, this work opens up new avenues for future research. We present several directions that are promising for tackling more complex scenarios and practical applications.

Our TAMP approach can be classified as a sequence-before-satisfy approach introduced in Section II, where abstract plans composed of a sequence of abstract actions are first found, followed by their refinement to create executable low-level motions. If the refinement of any abstract action fails, backtracking is commonly employed to revert to the previously computed low-level motions of the previous abstract action and attempt to find alternative low-level motions. However, the current MDP formulation does not allow for backtracking. If we achieve this capability, then we can expect the effort allocation strategy to effectively handle TAMP problems that may include many infeasible plans where no low-level motions exist.

Another important future direction is to consider various environments where the quantity, class, and shape of objects may vary, instead of a fixed environment. This approach requires representing planning time and execution time distributions that are generalizable to different environments. Generative models may be employed to learn these distributions, which can be conditioned on environment-specific features.

Taking into account exogenous processes in deadline-aware TAMP would allow for addressing a richer class of problems. For example, consider a scenario where one of the abstract actions involves standing in line to grab a coffee, but the waiting time is uncertain. Exogenous processes model uncertain events such as this uncertain waiting time, providing additional sources of uncertainty besides planning and execution times.

The long-term vision of this work is to create scalable, adaptive methods for solving TAMP

problems under deadline constraints. As we continue to refine our approach, integrating learning-based methods for generalizing across environments and handling more complex scenarios will be crucial in making deadline-aware TAMP a reliable tool for a wide range of robotic applications.

ACKNOWLEDGMENTS

REFERENCES

- [1] 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.
- [2] Z. Zhao, S. Chen, Y. Ding, Z. Zhou, S. Zhang, D. Xu, and Y. Zhao, “A survey of optimization-based task and motion planning: From classical to learning approaches,” *arXiv preprint arXiv:2404.02817*, 2024.
- [3] K. Hauser and V. Ng-Thow-Hing, “Randomized multi-modal motion planning for a humanoid robot manipulation task,” *The International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.
- [4] A. Krontiris and K. E. Bekris, “Dealing with difficult instances of object rearrangement.” in *Robotics: Science and Systems*, vol. 1123, 2015.
- [5] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, “Ffrob: Leveraging symbolic planning for efficient task and motion planning,” *The International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, 2018.
- [6] J. Wolfe, B. Marthi, and S. Russell, “Combined task and motion planning for mobile manipulation,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 20, 2010, pp. 254–257.
- [7] 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.
- [8] M. Toussaint, “Logic-geometric programming: An optimization-based approach to combined task and motion planning.” in *IJCAI*, 2015, pp. 1930–1936.
- [9] F. Lagriffoul and B. Andres, “Combining task and motion planning: A culprit detection problem,” *The International Journal of Robotics Research*, vol. 35, no. 8, pp. 890–927, 2016.

- [10] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, “An incremental constraint-based framework for task and motion planning,” *The International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.
- [11] S.-Y. Lo, S. Zhang, and P. Stone, “The petlon algorithm to plan efficiently for task-level-optimal navigation,” *Journal of Artificial Intelligence Research*, vol. 69, pp. 471–500, 2020.
- [12] 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.
- [13] Y. Sung, Z. Wang, and P. Stone, “Learning to correct mistakes: Backjumping in long-horizon task and motion planning,” in *Conference on Robot Learning*. PMLR, 2023, pp. 2115–2124.
- [14] 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.
- [15] L. P. Kaelbling and T. Lozano-Pérez, “Integrated task and motion planning in belief space,” *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1194–1227, 2013.
- [16] B. Kim, L. Shimanuki, L. P. Kaelbling, and T. Lozano-Pérez, “Representation, learning, and planning algorithms for geometric task and motion planning,” *The International Journal of Robotics Research*, vol. 41, no. 2, pp. 210–231, 2022.
- [17] Z. Kingston and L. E. Kavraki, “Scaling multimodal planning: Using experience and informing discrete search,” *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 128–146, 2022.
- [18] 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.
- [19] 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.
- [20] X. Fang, C. R. Garrett, C. Eppner, T. Lozano-Pérez, L. P. Kaelbling, and D. Fox, “Dimsam:

- Diffusion models as samplers for task and motion planning under partial observability,” *arXiv preprint arXiv:2306.13196*, 2023.
- [21] A. M. Wells, N. T. Dantam, A. Shrivastava, and L. E. Kavraki, “Learning feasibility for task and motion planning in tabletop environments,” *IEEE robotics and automation letters*, vol. 4, no. 2, pp. 1255–1262, 2019.
- [22] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint, “Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning,” in *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2020, pp. 9563–9569.
- [23] S. Li and N. T. Dantam, “A sampling and learning framework to prove motion planning infeasibility,” *The International Journal of Robotics Research*, vol. 42, no. 10, pp. 938–956, 2023.
- [24] Z. Yang, C. Garrett, T. Lozano-Perez, L. Kaelbling, and D. Fox, “Sequence-based plan feasibility prediction for efficient task and motion planning,” in *Robotics science and systems*, 2023.
- [25] Y. Sung and P. Stone, “Motion planning (in) feasibility detection using a prior roadmap via path and cut search,” *arXiv preprint arXiv:2305.10395*, 2023.
- [26] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez, “Learning compositional models of robot skills for task and motion planning,” *The International Journal of Robotics Research*, vol. 40, no. 6-7, pp. 866–894, 2021.
- [27] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez, “Learning symbolic operators for task and motion planning,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 3182–3189.
- [28] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, “Learning neuro-symbolic skills for bilevel planning,” *arXiv preprint arXiv:2206.10680*, 2022.
- [29] S. Cheng, C. Garrett, A. Mandlekar, and D. Xu, “Nod-tamp: Multi-step manipulation planning with neural object descriptors,” *arXiv preprint arXiv:2311.01530*, 2023.
- [30] U. A. Mishra, S. Xue, Y. Chen, and D. Xu, “Generative skill chaining: Long-horizon skill planning with diffusion models,” in *Conference on Robot Learning*. PMLR, 2023, pp. 2905–2925.
- [31] A. Curtis, X. Fang, L. P. Kaelbling, T. Lozano-Pérez, and C. R. Garrett, “Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances,”

- in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1940–1946.
- [32] Y. Ding, X. Zhang, X. Zhan, and S. Zhang, “Learning to ground objects for robot task and motion planning,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5536–5543, 2022.
- [33] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum, “Predicate invention for bilevel planning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, 2023, pp. 12 120–12 129.
- [34] W. Vega-Brown and N. Roy, “Asymptotically optimal planning under piecewise-analytic constraints,” in *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*. Springer, 2020, pp. 528–543.
- [35] R. Shome, D. Nakhimovich, and K. E. Bekris, “Pushing the boundaries of asymptotic optimality in integrated task and motion planning,” in *Algorithmic Foundations of Robotics XIV: Proceedings of the Fourteenth Workshop on the Algorithmic Foundations of Robotics 14*. Springer, 2021, pp. 467–484.
- [36] W. Thomason, M. P. Strub, and J. D. Gammell, “Task and motion informed trees (tmit*): Almost-surely asymptotically optimal integrated task and motion planning,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 370–11 377, 2022.
- [37] Y. Sung, Z. Chen, J. Das, P. Tokekar *et al.*, “A survey of decision-theoretic approaches for robotic environmental monitoring,” *Foundations and Trends® in Robotics*, vol. 11, no. 4, pp. 225–315, 2023.
- [38] E. Balas, “The prize collecting traveling salesman problem,” *Networks*, vol. 19, no. 6, pp. 621–636, 1989.
- [39] C. Chekuri and M. Pal, “A recursive greedy algorithm for walks in directed graphs,” in *46th annual IEEE symposium on foundations of computer science (FOCS’05)*. IEEE, 2005, pp. 245–253.
- [40] T. Lattimore and C. Szepesvári, *Bandit algorithms*. Cambridge University Press, 2020.
- [41] B. Liu, X. Xiao, and P. Stone, “Team orienteering coverage planning with uncertain reward,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 9728–9733.
- [42] S. J. Russell and E. Wefald, “Principles of metareasoning,” *Artificial Intelligence*, vol. 49,

- no. 1-3, pp. 361–395, 1991. [Online]. Available: [https://doi.org/10.1016/0004-3702\(91\)90015-C](https://doi.org/10.1016/0004-3702(91)90015-C)
- [43] T. L. Dean and M. S. Boddy, “An analysis of time-dependent planning.” in *AAAI*, vol. 88, 1988, pp. 49–54.
- [44] E. J. Horvitz, *Computation and action under bounded resources*. stanford university, 1991.
- [45] M. T. Cox, “Metacognition in computation: A selected research review,” *Artificial intelligence*, vol. 169, no. 2, pp. 104–141, 2005.
- [46] 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.
- [47] 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.
- [48] J. W. Herrmann, *Metareasoning for Robots: Adapting in Dynamic and Uncertain Environments*. Springer Nature, 2023.
- [49] 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.
- [50] 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.
- [51] 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.
- [52] M. Budd, B. Lacerda, and N. Hawes, “Stop! planner time: Metareasoning for probabilistic planning using learned performance profiles,” in *Proceedings of the 38th AAAI Conference on Artificial Intelligence (AAAI)*, Feb. 2024.
- [53] D. O’Ceallaigh and W. Ruml, “Metareasoning in real-time heuristic search,” in *Proceedings of the International Symposium on Combinatorial Search*, vol. 6, no. 1, 2015, pp. 87–95.
- [54] C. H. Lin, A. Kolobov, E. Kamar, and E. Horvitz, “Metareasoning for planning under uncertainty,” *arXiv preprint arXiv:1505.00399*, 2015.
- [55] Y. Sung, L. P. Kaelbling, and T. Lozano-Pérez, “Learning when to quit: meta-reasoning for motion planning,” in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 4692–4699.

- [56] J. Svegliato, K. H. Wray, S. J. Witwicki, J. Biswas, and S. Zilberstein, “Belief space metareasoning for exception recovery,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 1224–1229.
- [57] 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.
- [58] 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,” *Advances in neural information processing systems*, vol. 27, 2014.
- [59] F. Callaway, S. Gul, P. Krueger, T. Griffiths, and F. Lieder, “Learning to select computations,” in *34th Conference on Uncertainty in Artificial Intelligence (UAI 2018)*. Curran Associates, Inc., 2018, pp. 776–785.
- [60] S. S. Shperberg, A. Coles, B. Cserna, E. Karpas, W. Ruml, and S. E. Shimony, “Allocating planning effort when actions expire,” in *AAAI 2019*. AAAI Press, 2019, pp. 2371–2378.
- [61] M. Cashmore, A. Coles, B. Cserna, E. Karpas, D. Magazzeni, and W. Ruml, “Temporal planning while the clock ticks,” in *Proceedings of ICAPS*, 2018, pp. 39–46. [Online]. Available: <https://aaai.org/ocs/index.php/ICAPS/ICAPS18/paper/view/17724>
- [62] A. Elboher, A. Bensoussan, E. Karpas, W. Ruml, S. S. Shperberg, and S. E. Shimony, “A formal metareasoning model of concurrent planning and execution,” in *AAAI 2023*, 2023.
- [63] A. Coles, E. Karpas, A. Lavrinenko, W. Ruml, S. E. Shimony, and S. Shperberg, “Planning and acting while the clock ticks.” in *ICAPS*, 2024, pp. 232–239.
- [64] R. Diankov, “Automated construction of robotic manipulation programs,” Ph.D. dissertation, Carnegie Mellon University, USA, 2010.
- [65] T. Bylander, “The computational complexity of propositional strips planning,” *Artificial Intelligence*, vol. 69, no. 1-2, pp. 165–204, 1994.
- [66] J. H. Reif, “Complexity of the mover’s problem and generalizations,” in *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. IEEE Computer Society, 1979, pp. 421–427.
- [67] J. Canny, *The complexity of robot motion planning*. MIT press, 1988.
- [68] 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.

- [69] T. Lozano-Pérez and L. P. Kaelbling, “A constraint-based method for solving sequential manipulation planning problems,” in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 3684–3691.
- [70] M. Helmert, “The fast downward planning system,” *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.
- [71] D. Speck, R. Mattmüller, and B. Nebel, “Symbolic top-k planning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 06, 2020, pp. 9967–9974.
- [72] T. Ren, G. Chalvatzaki, and J. Peters, “Extended tree search for robot task and motion planning,” *arXiv preprint arXiv:2103.05456*, 2021.
- [73] B. Marthi, S. Russell, and J. A. Wolfe, “Angelic semantics for high-level actions.” in *ICAPS*, 2007, pp. 232–239.
- [74] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-completeness*. W. H. Freeman and Co., 1979, p. 190.
- [75] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, “A survey of monte carlo tree search methods,” *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.
- [76] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, “Monte carlo tree search: A review of recent modifications and applications,” *Artificial Intelligence Review*, vol. 56, no. 3, pp. 2497–2562, 2023.
- [77] L. Kocsis and C. Szepesvári, “Bandit based monte-carlo planning,” in *European conference on machine learning*. Springer, 2006, pp. 282–293.
- [78] S. S. Shperberg, A. Coles, E. Karpas, W. Ruml, and S. E. Shimony, “Situating temporal planning using deadline-aware metareasoning,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 31, 2021, pp. 340–348.
- [79] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [80] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven exploration by self-supervised prediction,” in *ICML*, ser. Proceedings of Machine Learning Research, vol. 70. PMLR, 2017, pp. 2778–2787.
- [81] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune, “Go-explore: a new

- approach for hard-exploration problems,” *CoRR*, vol. abs/1901.10995, 2019.
- [82] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation for games, robotics and machine learning,” 2016.
- [83] J. J. Kuffner and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001.
- [84] J. Achiam, “Spinning up in deep reinforcement learning,” 2018.

APPENDIX A

PLAN SKELETONS FOR THE MANIPULATION DOMAIN

We show the details of the plan skeletons for the manipulation domain. As numerous actions are involved in this domain, we combine `move` and `pick`, as well as `move` and `place`, into single abstract actions for simplicity.

- $\sigma_1 = (\delta_1^1 = \text{move} + \text{pick}(o_1 = B1), \delta_1^2 = \text{move} + \text{place}(o_1 = B1, o_f = R1), \delta_1^3 = \text{move} + \text{pick}(o_2 = B2), \delta_1^4 = \text{move} + \text{place}(o_2 = B2, o_f = R2))$,
- $\sigma_2 = (\delta_2^1 = \text{move} + \text{pick}(o_1 = B1), \delta_2^2 = \text{move} + \text{place}(o_1 = B1, o_f = R1), \delta_2^3 = \text{move} + \text{pick}(o_6 = B6), \delta_2^4 = \text{move} + \text{place}(o_6 = B6, o_f = C2), \delta_2^5 = \text{move} + \text{pick}(o_2 = B2), \delta_2^6 = \text{move} + \text{place}(o_2 = B2, o_f = R2))$,
- $\sigma_3 = (\delta_3^1 = \text{move} + \text{pick}(o_1 = B1), \delta_3^2 = \text{move} + \text{place}(o_1 = B1, o_f = R1), \delta_3^3 = \text{move} + \text{pick}(o_5 = B5), \delta_3^4 = \text{move} + \text{place}(o_5 = B5, o_f = C2), \delta_3^5 = \text{move} + \text{pick}(o_2 = B2), \delta_3^6 = \text{move} + \text{place}(o_2 = B2, o_f = R2))$,
- $\sigma_4 = (\delta_4^1 = \text{move} + \text{pick}(o_1 = B1), \delta_4^2 = \text{move} + \text{place}(o_1 = B1, o_f = R1), \delta_4^3 = \text{move} + \text{pick}(o_5 = B5), \delta_4^4 = \text{move} + \text{place}(o_5 = B5, o_f = C2), \delta_4^5 = \text{move} + \text{pick}(o_6 = B6), \delta_4^6 = \text{move} + \text{place}(o_6 = B6, o_f = C2), \delta_4^7 = \text{move} + \text{pick}(o_2 = B2), \delta_4^8 = \text{move} + \text{place}(o_2 = B2, o_f = R2))$,
- $\sigma_5 = (\delta_5^1 = \text{move} + \text{pick}(o_3 = B3), \delta_5^2 = \text{move} + \text{place}(o_3 = B3, o_f = C1), \delta_5^3 = \text{move} + \text{pick}(o_4 = B4), \delta_5^4 = \text{move} + \text{place}(o_4 = B4, o_f = C1), \delta_5^5 = \text{move} + \text{pick}(o_1 = B1), \delta_5^6 = \text{move} + \text{place}(o_1 = B1, o_f = R1), \delta_5^7 = \text{move} + \text{pick}(o_2 = B2), \delta_5^8 = \text{move} + \text{place}(o_2 = B2, o_f = R2))$,

- $\sigma_6 = (\delta_6^1 = \text{move} + \text{pick}(o_3 = B3), \delta_6^2 = \text{move} + \text{place}(o_3 = B3, o_f = C1), \delta_6^3 = \text{move} + \text{pick}(o_4 = B4), \delta_6^4 = \text{move} + \text{place}(o_4 = B4, o_f = C1), \delta_6^5 = \text{move} + \text{pick}(o_1 = B1), \delta_6^6 = \text{move} + \text{place}(o_1 = B1, o_f = R1), \delta_6^7 = \text{move} + \text{pick}(o_6 = B6), \delta_6^8 = \text{move} + \text{place}(o_6 = B6, o_f = C2), \delta_6^9 = \text{move} + \text{pick}(o_2 = B2), \delta_6^{10} = \text{move} + \text{place}(o_2 = B2, o_f = R2)),$
- $\sigma_7 = (\delta_7^1 = \text{move} + \text{pick}(o_3 = B3), \delta_7^2 = \text{move} + \text{place}(o_3 = B3, o_f = C1), \delta_7^3 = \text{move} + \text{pick}(o_4 = B4), \delta_7^4 = \text{move} + \text{place}(o_4 = B4, o_f = C1), \delta_7^5 = \text{move} + \text{pick}(o_1 = B1), \delta_7^6 = \text{move} + \text{place}(o_1 = B1, o_f = R1), \delta_7^7 = \text{move} + \text{pick}(o_5 = B5), \delta_7^8 = \text{move} + \text{place}(o_5 = B5, o_f = C2), \delta_7^9 = \text{move} + \text{pick}(o_2 = B2), \delta_7^{10} = \text{move} + \text{place}(o_2 = B2, o_f = R2)),$
- $\sigma_8 = (\delta_8^1 = \text{move} + \text{pick}(o_3 = B3), \delta_8^2 = \text{move} + \text{place}(o_3 = B3, o_f = C1), \delta_8^3 = \text{move} + \text{pick}(o_4 = B4), \delta_8^4 = \text{move} + \text{place}(o_4 = B4, o_f = C1), \delta_8^5 = \text{move} + \text{pick}(o_1 = B1), \delta_8^6 = \text{move} + \text{place}(o_1 = B1, o_f = R1), \delta_8^7 = \text{move} + \text{pick}(o_5 = B5), \delta_8^8 = \text{move} + \text{place}(o_5 = B5, o_f = C2), \delta_8^9 = \text{move} + \text{pick}(o_6 = B6), \delta_8^{10} = \text{move} + \text{place}(o_6 = B6, o_f = C2), \delta_8^{11} = \text{move} + \text{pick}(o_2 = B2), \delta_8^{12} = \text{move} + \text{place}(o_2 = B2, o_f = R2)).$

APPENDIX B

NETWORK ARCHITECTURE AND HYPERPARAMETER DETAILS FOR PPO

Two fully-connected neural networks are used separately for the actor and critic, each comprising three hidden layers of 64 neurons connected by Tanh activations.

The following lists the hyperparameter values we set for the experiments:

- Clipping parameter: 0.2,
- Gamma parameter (discount factor): 0.99,
- Actor learning rate: 0.0003,
- Critic learning rate: 0.001,
- Maximum time steps per episode: 1,000,
- Number of epochs at each iteration: 80,
- Time steps per epoch: 4,000.

APPENDIX C

A COLLECTION OF HISTOGRAMS FOR PLANNING AND EXECUTION TIMES

We present a collection of histograms for planning and execution times for all examples and experiments, each based on 1,000 trials.

Due to space limitations, the histograms are provided at the following link: <https://drive.google.com/file/d/1BPQeLd7E3QvLuwr7tyvoKnak3ClhQa91/view?usp=sharing>