2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Extended Tree Search for Robot Task and Motion Planning

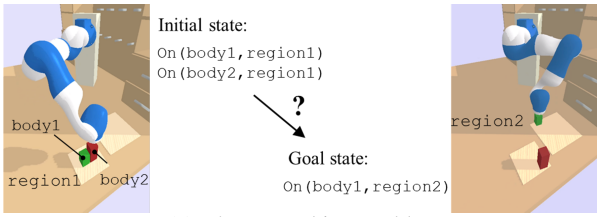
Tianyu Ren[1], Georgia Chalvatzaki[2], and Jan Peters[3]

***Abstract*— Integrated Task and Motion Planning (TAMP) offers opportunities for achieving generalized autonomy in robots but also poses challenges. It involves searching in both symbolic task space and high-dimensional motion space, while also addressing geometrically infeasible actions within its hierarchical process. We introduce a novel TAMP decision-making framework, utilizing an extended decision tree for both symbolic task planning and high-dimensional motion variable binding. Employing top-k planning, we generate a skeleton space with diverse candidate plans, seamlessly integrating it with motion variable spaces into an extended decision space. Subsequently, Monte-Carlo Tree Search (MCTS) is utilized to maintain a balance between exploration and exploitation at decision nodes, ultimately yielding optimal solutions. Our approach combines symbolic top-k planning with concrete motion variable binding, leveraging MCTS for proven optimality, resulting in a powerful algorithm for handling combinatorial complexity in long-horizon manipulation tasks. Empirical evaluations demonstrate the algorithm's effectiveness in diverse, challenging robot tasks, in comparison with the baseline methods.**

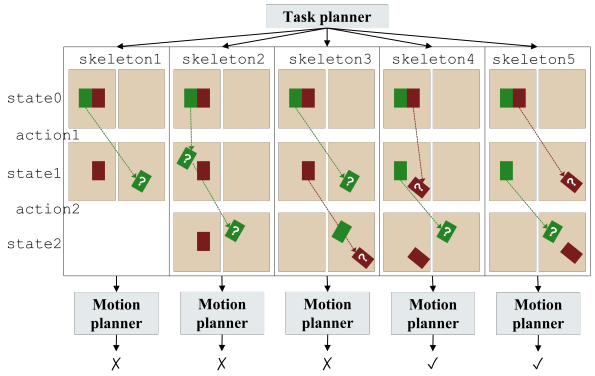
I. Introduction

Conventional methods, such as [1], historically approached long-horizon manipulation by strictly decomposing symbolic sub-tasks for independent resolution by motion planners. Nevertheless, in practice, significant dependencies exist between symbolic and geometric levels. Illustrated through a toy problem (Fig 1a) and its TAMP solutions (Fig 1b), the success of motion planners is contingent on task planner outputs, and vice versa. Efficient planning necessitates integration of both task and motion planning to ensure a coherent strategy. Our focus is on an integrated planner capable of navigating extensive action and motion spaces while addressing the challenge of infeasible skeletons resulting from their combination.

Infeasible task skeletons. The state abstraction inherent in TAMP hinders task planners from accessing detailed environmental constraints that may be crucial to problem solutions. In Fig,1a, for instance, the task planner lacks information from the symbolic initial state about the blockade of the taller body, potentially leading to the generation of `skeleton1` as the initial attempt. This challenge is commonly addressed in AI planning research as an *incomplete domain description* [3, 4], indicating insufficient domain knowledge for generating entirely accurate results. To mitigate this, the TAMP task planner must offer alternative skeletons and optimally select



(a) The Unpacking problem



(b) A hierarchical (or "search-then-sample" [2]) TAMP workflow

Fig. 1: The unpacking problem. The taller red body in the environment must be relocated before the green one can be reached without collision.

them for further examination based on the binding search report from motion planning.

This paper introduces an extended tree search for TAMP (*eTAMP*) to address the aforementioned challenges within a unified framework, employing a streamlined pipeline [1]. In the first step (Sec.,IV), the task planning problem is solved using a top-k planner, generating diverse high-level plans. These plans form the basis for constructing a skeleton space in the subsequent step. In the second step (Sec.,V), a decision tree is established, initiating with the first decision node for skeleton selection. From the second decision node onwards, a motion planner commences a binding search for each skeleton branch, creating an *extended decision tree* that synergizes high-level skeleton selection with low-level binding searches. Following a predefined number of roll-outs, eTAMP outputs the plan associated with the highest value on the tree branch. The eTAMP algorithm is summarized in Sec.,VI. The main contribution of this work is twofold:

- We propose a top-k skeleton planner based on generic top-k algorithms that generates diverse high-level plans in large task spaces.
- We introduce an extended decision tree that incorporates

[1]Tianyu Ren is with Computer Science Department, Technische Universität Darmstadt, Germany `tianyu@robot-learning.de`
[2]Georgia Chalvatzaki is with Computer Science Department, Technische Universität Darmstadt, Germany `georgia.chalvatzaki@tu-darmstadt.de`
[3]Jan Peters is with Computer Science Department, Technische Universität Darmstadt, Germany `jan.peters@tu-darmstadt.de`

[1]The code is available online at https://github.com/ttianyuren/eTAMP

both skeleton selection under infeasible task skeletons and binding search in large motion spaces within a unified optimal decision process.

We evaluate the proposed method on various robotic tasks based on the TAMP benchmark domains proposed in [5], using both a 7-degrees-of-freedom (dof) robot and a 10-dof mobile manipulator. Our empirical results prove the ability of eTAMP to find feasible plans across a spectrum of challenging scenarios.

II. Related Work

TAMP encompasses multi-modal motion planning challenges, commonly approached as optimization problems through logic-geometric programming [6], or multi-modal motion planning with motion-modes switches for various tasks [7, 8, 9, 10]. However, these methods are designed for particular robot domains, e.g., desktop manipulation [6], in-hand manipulation [9], and humanoid locomotion [8]. Individual tasks need to be manually factorized by constraint graphs before planning [10]. They apply to neither problems with large task spaces such as the game of Hanoi Tower nor those with non-geometric actions. On the other hand, general-purpose TAMP systems generically use symbolic AI planners for efficient problem-solving in task spaces, where the relational language PDDL is used for the description of the symbolic domain and problem. A comprehensive review of integrated TAMP approaches can be found in [2]. Here we discuss only those that are closely related to our method.

In early TAMP approaches like *aSyMov* [11] and *SMAP* [12], the framework exhibited a unidirectional flow from task planning to motion planning. While symbolic action plans effectively guided high-dimensional motion planning through established links between symbolic and geometric descriptions, these systems became less efficient when problems lacked sufficient symbolic constraints, resulting in infeasible skeletons. Approaches like *FFRob* [13] and *PDDLStream* [14, 15] addressed this by allowing action planners to validate preconditions in the geometric space during action plan computation, strengthening task-motion interplay. However, these methods require careful crafting of symbolic descriptions, including geometric constraints in all high-level actions.

In the absence of explicit geometric constraints, the TAMP system anticipates more infeasible skeletons and a larger motion space for exploration. As the feasibility of a skeleton is assessed through motion planning, the task planner needs to incorporate feedback from motion planning results for adaptive decision-making. Approaches proposed in [16], [17], and [18] iteratively refine high-level representations based on low-level planning feedback during motion binding searches. Failures in motion planning are symbolized into logical predicates, updating high-level models, and enabling the generation of new skeletons that may circumvent geometric failures. Due to challenges in symbolizing continuous geometric failures in high-level models, these methods often resort to discretizing certain dimensions of motion spaces, such as using discrete variables for object positions in robot tasks [17].

This study builds upon recent advancements in Task and Motion Planning (TAMP) and generic AI planning. We introduce a novel method aimed at integrating search in extensive task and motion spaces while minimizing human inputs. In contrast to previous TAMP approaches, our method explicitly models the skeleton selection process, achieving optimal selection through (i) maintaining an explicit skeleton space and (ii) consistently evaluating the value of each skeleton. We generate a skeleton space comprising diverse high-level plans using a top-k planner. Top-k planning involves finding a set of diverse solutions of size k, offering a range of candidate skeletons to navigate around infeasible options. Based on *FastDownward* [19, 20], [21] presents a complete top-k planner *SYM-K* that scales efficiently to large sizes of k. We estimate the value of each skeleton by employing the back-propagation mechanism of Monte Carlo Tree Search (MCTS) during the binding search. Specifically, we use *PW-UCT* (Upper Confidence bounds for Trees with Progressive Widening) [22, 23, 24], to solve the sequential binding problem over large motion spaces. This tree search structure seamlessly integrates discrete skeleton selection with continuous motion planning.

III. Preliminaries

PDDLStream description.

PDDLStream [14, 15] enhances PDDL universality by integrating *streams* into PDDL operators alongside actions. A PDDLStream task, denoted as $T_{stream} = \langle Objects, S, G, Actions, Streams \rangle$, comprises a finite set of objects ($Objects$) associated with the task, initial and goal states (S and G), actions ($Actions$) for environment updates, and streams ($Streams$) serving as black-box generators for new objects, such as robot paths or grasp directions.

A skeleton π_s composed of both actions and streams can be found as

$$\begin{aligned}&\langle \texttt{Sample-pose(body1, region2)} \rightarrow \texttt{\#poseA},\\ &\texttt{Plan-motion(body1, pose1, \#poseA)} \rightarrow \texttt{\#trajA},\\ &\texttt{Pick-place(body1, region1, region2, \#trajA)} \rangle.\end{aligned} \tag{1}$$

Objects marked by $\#$ (e.g., `#poseA`) are generated by streams and they require concrete *bindings* during the binding search. We denote this generation process as $\#Objects, \#S = \text{OPTMS-EXPD}(Objects, S, Streams, level)$. For detailed implementation, refer to [15].

IV. Top-k skeleton planning

In task planning, we generate diverse skeletons via top-k planning. Let P_T be the set of all symbolic plans (possibly infinite) for a planning task T. The objective of top-k planning is to determine a set of k different plans $\{\pi_1, \pi_2, ..., \pi_k\} = P \subseteq P_T$ with the lowest costs for a given planning task [21]. We call a top-k algorithm *complete* iff it can find the

set of all plans as required. A standard top-k planner can be described as $P = \text{TOP-K}(Objects, S, G, Operators, k)$.

A. Search for top-k plans, without steams

The planner firstly searches for the top-k plans P_a based on the optimistically enriched objects $\#Objects$ and initial state $\#S$. Involved operators are solely actions.

$$P_a = \text{TOP-K}(\#Objects, \#S, G, Actions, k) \quad (2)$$

B. Search for top-k skeletons

Then the streams that generate those involved optimistic objects $\#Objects \setminus Objects$ are retracted and integrated into the action plans. Finally, we have the top-k skeletons P_s which are composed of both actions and streams. Each of these skeletons only relies on the original objects $Objects$ and initial state S.

$$\pi_s = \text{TOP-K}(Objects, S, G_s(\pi_a, G), Actions \cup Streams, 1) \quad (3)$$

$Actions$ and $Streams$ constitute the operator set of this planning problem. Informally, by setting $k = 1$, the top-k algorithm reduces to a classical PDDL planner. To ensure that the sequences of actions maintain from π_a to π_s, we add extra constraints to the goal state. By assuming $\pi_a = \langle a_1, a_2, ..., a_n \rangle$, we define the updated goal state G_s as below. $(a_1 \prec a_2)$ is a literal asserting that a_1 is the predecessor of a_2.

$$G_s(\pi_a, G) = \{(a_1 \prec a_2), ..., (a_{n-1} \prec a_n)\} \cup G, \quad (4)$$

C. Algorithm summary

Algorithm 1: TOP-K-SKELETON

Input: $Objs, S, G, Actions, Streams, k$
for $level \in [0, 1, ..., max\text{-}level]$ **do**
  $\#Objs, \#S =$ OPTMS-EXPD$(Objs, S, Streams, level)$,
  $P_a = \text{TOP-K}(\#Objs, \#S, G, Actions, k)$, see (2)
  if $|P_a| < k$ **then**
    continue for
  $P_s = \{\}$
  for $\pi_a \in P_a$ **do**
    $Operators = Actions \cup Streams$
    $\#G = G_s(\pi_a, G)$, see (4)
    $\pi_s = \text{TOP-K}(Objs, S, \#G, Operators, 1)$, see (3)
    $P_s \leftarrow \{\pi_s\} \cup P_s$
  return P_s

An overview of the above skeleton planning algorithm is shown in Alg. 1. The inputs of the algorithm include the original initial state S, goal state G, available $Streams$ and $Actions$, and a desired number of skeletons k. The output P_s is the list of the top k skeletons. The size of $\#Objects$ ramps up quickly with increasing $level$. To address this, in Alg. 1 we regulate the number of optimistic objects by progressively increasing $level$. Here we choose the SYM-K algorithm as our top-k planning subroutines in (2) and (3) due to its proven completeness and soundness [21]. Accordingly, we can derive the completeness property of our top-k skeleton planning algorithm.

Property 1: Probabilistic completeness in the top-k skeleton planning. Given a complete top-k planner, any potentially feasible skeleton will be contained by the result set of Alg. 1 as k goes to infinity.

After applying Alg. 1 to the toy planning problem task (Fig. 1a), we get a feasible skeleton as shown in Table I.

TABLE I: An example skeleton to the toy problem in Fig. 1a with its operators grouped by tree node layers.

Layers	Operators
decision1	`Sample-pose(body2,region1)` → `#poseB`
transition1	`Plan-motion(body2,pose2,#poseB)` → `#trajB` `Pick-place(body2,region1,region1,#trajB)`
decision2	`Sample-pose(body1,region2)` → `#poseC`
transition2	`Plan-motion(body1,pose1,#poseC)` → `#trajC` `Pick-place(body1,region1,region2,#trajC)`

V. Tree Search in the Extended Decision Space

Based on the top-k skeletons generated above, we must search for feasible concrete plans that are geometrically feasible and further find the optimal plan with the highest reward. To achieve this, firstly, we have to choose from the candidate skeletons (*skeleton space*) the one that we would like to continue with in binding search (Sec. V-B); secondly, we must search for concrete bindings of its symbolic variables (e.g., `#pose12` in (1)) in a *motion space* (Sec. V-A). In this study, we define a reward function (5) as the optimization target for both skeleton selection and binding search. For a skeleton π_i with H_i symbolic variables to bind (*binding horizon*), we have

$$r = p_t \left(\frac{1 + d_{\text{end}}}{H_i} + \frac{p_m}{motionCost + 1} \right) + r_{\text{end}}. \quad (5)$$

d_{end} is the number of symbolic variables that have already found their feasible bindings and $motionCost$ is proportional to the swept volume of the robot from its initial state to the terminal state. The first term in (5) encourages the planner to avoid choices where fewer bindings are found. The second term makes the planner prefer choices with motions that have less occupation in the workspace. For the third term, $r_{\text{end}} = 1$ when all bindings are successfully found, otherwise $r_{\text{end}} = 0$. For the hyperparameters, we set $p_t = 0.1$ and $p_m = 1.0$.

A. Optimal search in the binding space

A skeleton logically outlines a path to the goal state, but its geometric feasibility is assessed through the sequential binding of symbolic variables to concrete values. The binding search problem is formulated as a Markov Decision Process (MDP), treated as a finite-horizon stochastic optimal search problem.

We propose using PW-UCT to systematically explore the decision space, addressing the mixed-motion space with both discrete and continuous decisions. PW-UCT, a variant of UCT with progressive widening techniques [22, 23, 24], limits visits to existing nodes artificially. Once the value of existing nodes is adequately estimated, new nodes are expanded to explore unvisited regions. At a decision node z, if PW-TEST(z) is true, a new child (or binding decision) of z is sampled, expanding the tree with a new branch; otherwise, one of its existing children is selected by UCB.

$$\text{PW-TEST}(z) = \lfloor \text{visits}(z)^{\alpha} \rfloor > \lfloor (\text{visits}(z) - 1)^{\alpha} \rfloor \quad (6)$$

In (6) index α is the corresponding PW constant for balancing expansion with evaluation [23]. The PW-UCT search tree is structured with interleaved layers of decision nodes and transition nodes. Skeletons are grouped into these two node types, as shown in Table.,I. Decision nodes handle independent bindings (e.g., `#pose21a` in **decision1**), while transition nodes handle dependent bindings (e.g., `#traj211a` in **transition1**) and update the environment state (e.g., `Pick-place` from **transition1**).

B. Optimal search in the skeleton space

We model skeleton selection as a multi-armed bandit problem with the reward function of (5), and solve it via UCB1 [25]. At beginning, we have $d_{end} = 0$, $r_{\text{end}} = 0$, and $motionCost = 0$ in 5. As binding search results become available, the reward function incorporates multiple factors, prioritizing skeletons with proven geometric feasibility (third term), a high percentage of found bindings (first term), and low motion cost (second term).

To ensure a complete search in task planning, a sufficiently large k in TOP-K-SKELETON (Alg.,1) is necessary. As $k \to \infty$, we utilize a continuum-armed bandit instead of a multi-arm one to model the extended root. Here, the progressive widening trick is applied again at the decision node (e_root) for skeleton selection. If PW-TEST(e_root) is true, a new skeleton is added to the skeleton space P_s; otherwise, the bandit continues evaluating existing skeletons.

C. The extended decision tree

Using a given skeleton, a decision tree for binding search is constructed, as depicted in the lower part of Fig.,2. The motion planner employs PW-UCT to search for bindings at each decision node (e.g., **d1**). At each step, a symbolic variable (e.g., `#pose21a`) is sampled, and the simulator is updated via a transition node (e.g., **t1.1**). If the resulting state is geometrically feasible (e.g., **state1.1**), the planner proceeds to bind the next symbolic variable (e.g., `#pose12a`). Otherwise, an unsuccessful binding decision results in a geometrically infeasible state (e.g., **state1.2**), terminating the current branch. Upon creating a terminate node, its reward propagates through its parents back to the root of the current skeleton branch [26].

Thus, the value of selecting a specific skeleton (e.g., `skeleton4`) is updated (orange arrows in Fig.,2), and it

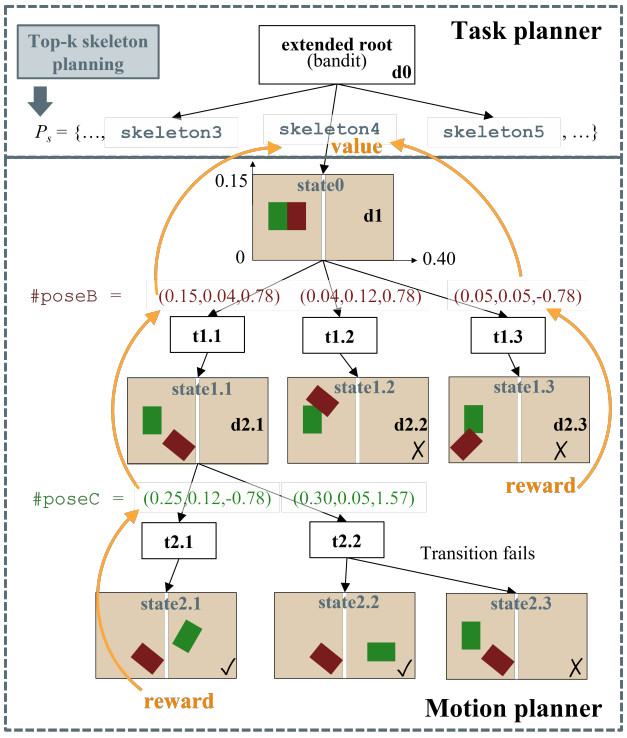


Fig. 2: The extended decision tree: an integration of optimal skeleton selection and binding search.

can be used by the multi-arm bandit in skeleton space (the upper part in Fig. 2). Examining Fig.,2 further reveals that the skeleton selection process is essentially an extended decision node before bind search, seamlessly integrated into the UCT structure. By establishing the bandit of skeleton selection as an extended root, we define an extended decision tree capable of searching in both the skeleton space and the motion space.

D. Algorithm summary

We can summarize the extended tree search algorithm in Alg. 2. The children of the extended root node e_root that represents the skeleton space are initialized with the empty set.

Alg.,2 takes top-k skeletons P_s from Alg.,1 as new children of the extended root, along with a user-defined time budget t_{ts}. It outputs a single concrete plan π_c, representing the best child of the extended root in terms of branch value. During ADD-VISIT($node$), $node$ increments its visit count by 1, and the environment resets to the initial state of $node$. UCB and PW criteria ensure a consistent value estimation for each node (e.g., **d1** in Fig.,2).

Property 2: Consistent value estimation in extend tree search. The value estimation of each tree node will converge to its real value as the search time goes infinite.

The property is evident through recursive analysis from leaf nodes to the root; detailed proof can be found in [23]. Consistent estimation of decision value ensures the global convergence of the output π_c to the optimal plan.

Property 3: Probabilistic completeness of extended tree search. The optimal concrete plan within the given skeleton space (implied by P_s) can be found after t_{ts} goes infinite.

Algorithm 2: EXTENDED-TREE-SEARCH

Input: P_s, t_{ts}
$new\_skeletons = P_s - e\_root$.CHILDREN
$e\_root = e\_root$.ADD-CHILDREN($new\_skeletons$)
$node \leftarrow e\_root$
while $timeCost() < t_{ts}$ **do**
  while $node$ *is not terminated* **do**
    ADD-VISIT($node$)
    if *PW-TEST*($node$) **then**
      $node \leftarrow$ EXPAND-NEW-CHILD($node$),
    else
      $node \leftarrow$ SELECT-CHILD-BY-UCB($node$),
  BACK-RPOPAGATE($node$)
  $node \leftarrow e\_root$
$\pi_c = e\_root$.BEST-CHILD
return π_c

VI. The eTAMP algorithm

We introduce some techniques to enhance the proposed method's efficiency in practice (Sec.,VI-A) and summarize the final algorithm (Sec.,VI-B).

A. Incremental skeleton space

Theoretically, a TAMP framework, as depicted in Fig.,2, can explore all combinations of skeletons and their bindings when the skeleton space P_s is exhaustive. However, even for state-of-the-art top-k planners like sym-k, a large k often leads to significantly prolonged planning times. To balance efficiency with completeness in task planning, we propose incrementally expanding the skeleton during the search. We divide the entire TAMP into sessions with incremental skeletons, where each session involves adding a batch of new skeletons to the skeleton space. The TAMP system searches the current skeleton space for a given time before transitioning to the next session.

B. Algorithm summary

Algorithm 3: eTAMP

Input: $T_{stream}, k_{batch}, t_{ts}, t_d$
$i = 1$
$P_s = \{\}$
e_root.CHILDREN $= \{\}$
while $timeCost() < t_d$ **do**
  $k = k_{batch} \cdot i$
  $P_s = P_s \,|\,$ TOP-SKELETON(T_{stream}, k), see Alg. 1
  $\pi_c =$ EXTENDED-TREE-SEARCH(P_s, t_{ts}), see Alg. 2
  $i = i + 1$
return π_c

A serial combination of Alg. 1 and Alg. 2 gives out the overall eTAMP algorithm, as summarized in Alg. 3. It takes as input a PDDLStream problem description $T_{stream} = \langle Objects, S, G, Actions, Streams \rangle$, and three hyper-parameters: k_{batch}, t_{ts} and t_b. k_{batch} is the number of new skeletons added to the skeleton space in the next extended tree search session, and t_{ts} is the time cost allocated to each session. t_b limits the total planning time of eTAMP. Given sufficiently large t_b, k in Alg. 3 will grow to infinity, and the extended root will have all possible skeletons as its children (*property 1*). Meanwhile, as the visit number of the extended root increases, the resultant concrete plan π_c will converge to the optimal one with respect to the reward function (5) (*property 3*).

TABLE II: The difficulties reflected by the evaluation domains

Difficulty	Kitchen	Tower	Unpacking	Regrasping
Infeasible task actions			✓	✓
Large task space		✓		
Large motion space*	✓	✓		✓
Non-monotonicity				✓
Non-geometric actions	✓			✓

*An extra criterion in addition to those of [5].

VII. Empirical evaluation

We empirically evaluate the eTAMP algorithm in four domains: Kitchen, Hanoi Tower, Unpacking, and Re-grasping. Kitchen and Hanoi Tower are adapted from the benchmark for TAMP systems proposed by [5], while the other two are based on our lab's existing robot hardware. The recognizable difficulties for each domain are listed in Table,II, measuring the planner's performance. In addition to criteria from [5], we include a new dimension, *large motion spaces*, to gauge the difficulty of constraint satisfaction in binding search. This criterion is satisfied if the underlying motion planning problem requires substantial search effort, applying to tasks with numerous bindings or tight feasible spaces for some bindings.

The study of PDDLStream is most similar to our work in terms of the domain description. Therefore, we use the *Adaptive* algorithm from [14], representing the best-performing PDDLStream method, as a baseline for evaluating eTAMP. Moreover, we compare eTAMP to *IDTMP* from [17] by partially discretizing the motion space. We choose the Unpacking domain for discretization as it is the only domain that does not feature large motion spaces in Table. II.

All TAMP algorithms use the same internal simulator implemented with PyBullet [27] for hosting the environment state. Planners are terminated after a 300-second timeout for each of the three tasks. In eTAMP, we set $k_{batch} = 50, t_{ts} = 250$. In the extended tree search (see Alg.,2), the MCTS rollout policy is random without heuristics guiding the search. PW-TEST (see (6)) takes $\alpha = 0.26$ to regulate the growth of new branches. Each algorithm's performance is evaluated over 100 instances for each task, and computation time is calculated when a feasible concrete plan is found. In the spirit of domain-independent planning, all planners maintain the same hyper-parameters across all domains. Notably, all planners use the same set of random stream generators for motion bindings, which have zero heuristics.

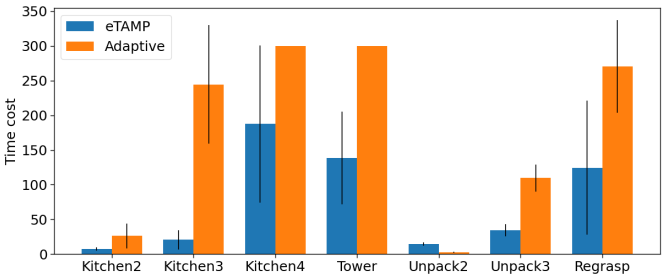


Fig. 3: The average time cost with std indicators of eTAMP and PDDLStream Adaptive [14] in solving the evaluation tasks.

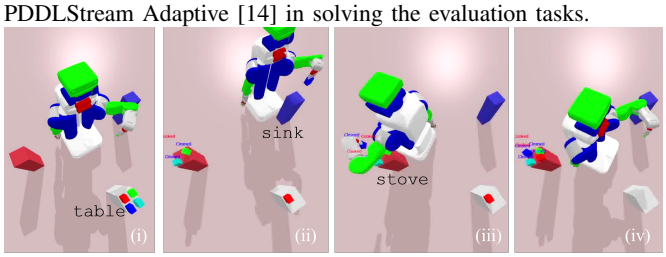


Fig. 4: In the Kitchen domain, the mobile manipulator must cook bodies by initially placing them on the sink for cleaning and subsequently placing them on the stove for cooking.

A. Kitchen domain

Compared to the original Kitchen task in [5], this domain reduces the complexity in task planning but meanwhile becomes more demanding to the motion planner. As shown in Fig. 4, the mobile manipulator must "cook" a given number of food blocks initially placed on `table`. Food must be cleaned before it can be cooked. It can be cleaned when placed on `sink` and cooked when placed on `stove`. Metric decisions for body pose are left to the motion planner for each placement. Actions of `Clean` (via `sink`) and `Cook` (via `stove`) are non-monotonic from the robot's perspective. The primary challenge lies in the tight stove region, where several bodies need placement. Feasible pose search for each new body becomes exponentially challenging after each placement. This domain assesses planners' performance in a *large motion space*. Without prior symbolic knowledge of potential congestion, planners must navigate trial and error to find feasible decisions. Consistent value estimation during binding search is critical for this delayed-reward scenario.

Evaluation results in Fig.,3 depict Kitchen2, Kitchen3, and Kitchen4 settings with 2, 3, and 4 bodies to be cooked, respectively. eTAMP outperforms Adaptive in scalability, showing potential problem-solving capabilities in Kitchen4. Specifically, for successful rates, both methods achieved 100% in Kitchen2. In Kitchen3 and Kitchen4, eTAMP achieved 100% and 60%, while Adaptive achieved 36% and 0%.

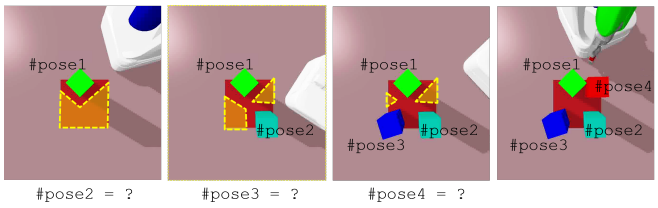


Fig. 5: The *large motion space* of the Kitchen domain. The feasible space (the yellow zone) for the pose of coming bodies shrinks significantly with each placement. A binding search for these poses is challenging given zero heuristics.

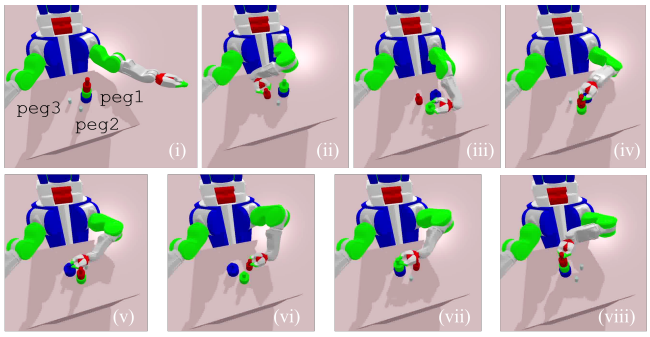


Fig. 6: Hanoi Tower domain. The robot must move all discs from `peg1` to `peg3` without causing any collision in motion spaces while complying with the game rule in task spaces.

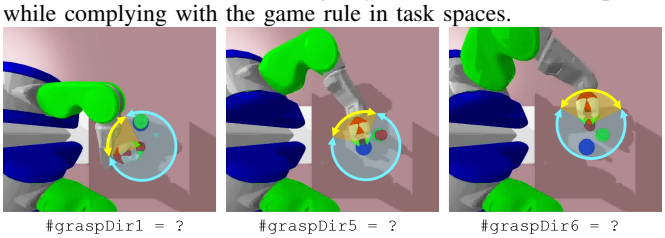


Fig. 7: The *large motion space* of the Hanoi Tower domain. The feasible grasp direction (the yellow zone) is small compared to the whole decision space. A binding search for these poses is challenging given zero heuristics. The motion planner has to bind `#graspDir1`,`#graspDir2`,...,`#graspDir8` to ground values (degrees) for the task completion.

B. Hanoi Tower domain

In this Hanoi Tower variant, only one arm of the PR2 robot is permitted for manipulation (Fig.,6). The task involves relocating a stack of three discs with different sizes from an initial pole to a target pole, passing through an intermediate pole. The game rule dictates that the upper disc must always be smaller than the lower one. Following the generation of a skeleton in the *large task space*, the motion planner faces the challenge of conducting a binding search for the grasping direction of each pick-place operation. Due to the confined workspace and joint limitations, the robot must meticulously explore this *large motion space*.

As shown in Fig. 3, eTAMP got a successful rate of 99% in the evaluation while Adaptive got 0% and could not give a solution within the time limitation. We attribute the poor performance of Adaptive in this domain to the tight decision space in binding search. As shown in Fig.,7, the feasible binding area is small compared to the large search space ($[-\pi, \pi]$ in this example) due to the obstruction of nearby entities and the kinematic configuration of the robot arm. The robot has to make eight successive decisions like this to complete the task, making it particularly challenging for TAMP methods not specifically designed to optimize the exploitation-exploration behavior in motion planning, such as Adaptive.

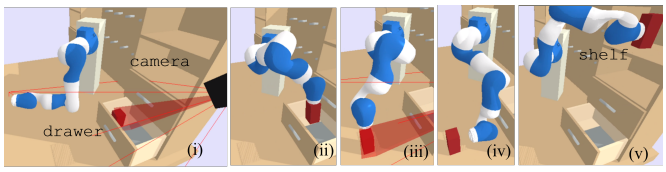


Fig. 8: Regrasping domain. The only way the robot can move the cuboid body from the drawer to the shelf is by re-grasping it in a different direction after taking it from the drawer.

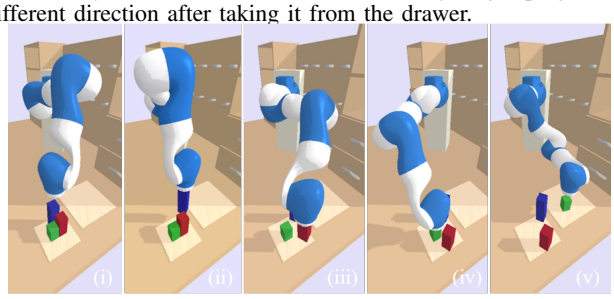


Fig. 9: The 3-body Unpacking task. To reach the green body, taller bodies must be relocated by the robot at first.

C. Regrasping domain

As depicted in Fig.,8, the robot must move the red block from the drawer to the shelf while keeping it upright. After each displacement, the block's location must be measured by a camera using the `locate-object` non-geometric action. Due to geometric constraints, the robot cannot move the block directly to its target area without re-grasping it from a different direction. The robot can choose from five directions to grasp the block: along the normal vectors of the top surface and four side surfaces. The block, initially positioned for top grasping in the drawer, must be grasped from the side when placed on the shelf (Fig.,8). This domain is challenging for task planning as it requires composing re-grasping behavior, and it is geometrically complex, necessitating a search in the hybrid decision space (discrete grasping directions and continuous body poses) for bindings (*large motion space*).

Fig. 3 shows eTAMP in this challenging task performs better than Adaptive. eTAMP presented a successful rate of 86% compared to that of 22% from Adaptive.

D. Unpacking domain

The task is derived from the motivation example in Fig.,1. Involving more than one body makes the problem challenging, as task planners lack information about geometric constraints, as exemplified by the taller red body in Fig.,1a (infeasible task actions). Testing TAMP algorithms in both a 2-body scenario (Unpack2) and a 3-body scenario (Unpack3) depicted in Fig.,9, where two taller bodies must be relocated before reaching the target one.

Adaptive and eTAMP achieved a 100% success rate for Unpack2 and Unpack3. Examining the details in Fig.,3, the methods exhibited distinct characteristics. In Unpack2, where motion planning is relatively simple, eTAMP takes more time than Adaptive as it generates top-k skeletons, many of which are unnecessary. In contrast, in Unpack3, with a larger motion planning space (though not as large as other evaluation domains), eTAMP demonstrated better performance due to its more efficient binding search.

E. Discrete Unpacking domain

While IDTMP is not directly applicable in all the mentioned domains, we included it as a baseline method to evaluate eTAMP in discrete motion spaces by discretizing `region1` and `region2` using 3×3 grids in the Unpack3 task. eTAMP solved the discrete task with an average time cost of $t_{average} = 109.45$ and $std = 4.30$, while IDTMP achieved $t_{average} = 82.00$ and $std = 44.39$. It is not surprising that IDTMP performs comparably or slightly better than eTAMP in this setting, as it directly learns geometric constraints from discrete binding errors to refine task plans. On the other hand, eTAMP is designed for large continuous motion spaces, making it challenging to weigh a single binding error as much as IDTMP does.

Another interesting observation from the comparison between discrete eTAMP and continuous eTAMP (Fig.,3) for the same task is that eTAMP shows less efficiency when the binding space is discretized, and less decision freedom is left to the planner. In fact, any attempt to discretize or reduce the motion spaces risks compromising the completeness of the planner. Therefore, having a TAMP system that can handle large motion spaces systematically is favorable.

F. Summary

In the above experiments, we evaluate the time cost of the considered TAMP algorithms for finding the first feasible plans. The optimality (e.g., motion cost) of these plans is not considered in this study. Evaluation results show the competitive performance of eTAMP in solving challenging TAMP problems as summarized in [5]. We have to admit that most state-of-the-art TAMP methods such as Adaptive are satisfying in solving tasks with *infeasible task actions* and *large task space*, as shown in the Unpack domain in Fig. 3. On the other hand, the unique tree search in the extended decision space makes eTAMP especially efficient when the motion space gets large and existing binding search practices start to suffer, which is proved by the experiments with the Kitchen domain, the Hanoi Tower domain, and the Re-grasping domain.

VIII. Conclusion

We propose eTAMP as a general-purpose planner of robot manipulation tasks with long horizons that demand symbolic sequencing of operators and binding search of motion parameters under geometric constraints. Similar to other general-purpose TAMP frameworks, eTAMP relies on its internal symbolic planner to deal with the *large task space* and *non-geometric actions*. What makes it unique is that it derives its capability of addressing the *infeasible task actions* from the diverse backup skeletons generated by a top-k planner and an optimal selection process of those skeletons. By modeling the binding search as MDPs, eTAMP utilizes PW-UCT to find ground values for the symbolic variables in skeletons and thus it is more prepared for the *large motion space* that is ubiquitous in realistic robotic manipulation.

REFERENCES

[1] R. E. Fikes and N. J. Nilsson, "STRIPS, a retrospective," *Artificial intelligence in perspective*, vol. 227, 1994.

[2] 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.

[3] C. Weber and D. Bryce, "Planning and acting in incomplete domains," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 21, 2011.

[4] H. H. Zhuo, T. A. Nguyen, and S. Kambhampati, "Refining Incomplete Planning Domain Models Through Plan Traces," in *IJCAI*, pp. 2451–2458, 2013.

[5] F. Lagriffoul, N. T. Dantam, C. Garrett, A. Akbari, S. Srivastava, and L. E. Kavraki, "Platform-independent benchmarks for task and motion planning," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3765–3772, 2018.

[6] M. Toussaint, "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning.," in *IJCAI*, pp. 1930–1936, 2015.

[7] Z. Kingston, A. M. Wells, M. Moll, and L. E. Kavraki, "Informing multi-modal planning with synergistic discrete leads," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3199–3205, IEEE, 2020.

[8] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, pp. 1–8, 2012.

[9] I. Mordatch, Z. Popović, and E. Todorov, "Contact-invariant optimization for hand manipulation," in *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*, pp. 137–144, 2012.

[10] J. Ortiz-Haro, V. N. Hartmann, O. S. Oguz, and M. Toussaint, "Learning efficient constraint graph sampling for robotic sequential manipulation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4606–4612, IEEE, 2021.

[11] S. Cambon, R. Alami, and F. Gravot, "A hybrid approach to intricate motion, manipulation and task planning," *The International Journal of Robotics Research*, vol. 28, no. 1, pp. 104–126, 2009.

[12] 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*, pp. 5002–5008, IEEE, 2010.

[13] 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.

[14] 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, pp. 440–448, 2020.

[15] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Stripstream: Integrating symbolic planners and blackbox samplers," *arXiv preprint arXiv:1802.08705*, 2018.

[16] 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)*, pp. 639–646, IEEE, 2014.

[17] 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.

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

[19] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[20] A. Torralba, V. Alcázar, D. Borrajo, P. Kissmann, and S. Edelkamp, "SymBA*: A symbolic bidirectional A* planner," in *International Planning Competition*, pp. 105–108, 2014.

[21] D. Speck, R. Mattmüller, and B. Nebel, "Symbolic top-k planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 9967–9974, 2020.

[22] R. Coulom, ""elo ratings" of move patterns in the game of go," *ICGA journal*, vol. 30, no. 4, pp. 198–208, 2007.

[23] D. Auger, A. Couetoux, and O. Teytaud, "Continuous upper confidence trees with polynomial exploration–consistency," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 194–209, Springer, 2013.

[24] G. M. J. Chaslot, M. H. Winands, H. J. V. D. HERIK, J. W. Uiterwijk, and B. Bouzy, "Progressive strategies for Monte-Carlo tree search," *New Mathematics and Natural Computation*, vol. 4, no. 03, pp. 343–357, 2008.

[25] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2, pp. 235–256, 2002.

[26] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *International conference on computers and games*, pp. 72–83, Springer, 2006.

[27] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," *URL http://pybullet. org*.