Task and Motion Planning Using Infinite Completion Tree and Agnostic Skills

Matan Sudry, Tom Jurgenson and Erez Karpas

Technion — Israel Institute of Technology, Haifa, Israel
{matansudry, tomj}@campus.technion.ac.il, karpase@technion.ac.il

Abstract

This work builds upon existing task and motion planning (TAMP) frameworks by integrating pre-trained Sequencing Task-Agnostic Policies (STAP) and an infinite completion tree to create a hierarchical approach that decouples high-level task decisions from low-level motion execution. The method enhances the planning process by incorporating a more accurate mean success rate estimator for task success predictions than traditional Q-value estimators. We formalize the problem of long-horizon manipulation tasks, where high-level decisions are made in discrete spaces and low-level actions are executed in continuous space. To guide the search process efficiently, we leverage the infinite completion tree, which dynamically adjusts computational resources based on task complexity. Empirical results demonstrate that our approach significantly improves planning efficiency and execution reliability, outperforming traditional methods by reducing the search space and computational overhead. Our work highlights the effectiveness of combining learned skills from STAP with infinite completion trees and accurate success predictors in a hierarchical structure, laying the foundation for scalable robotic planning in complex, real-world manipulation tasks.

Introduction

Sequential manipulation tasks in robotics are inherently complex, requiring reasoning about interactions between actions and objects in dynamic environments. Traditional approaches primarily focus on geometric feasibility, often relying on motion planning techniques (LaValle 2006; Toussaint 2015). However, these methods struggle with long-horizon planning because they fail to account for the relationship between high-level task objectives and low-level motion feasibility.

In our framework, an agent interacts with rigid objects in continuous space, where the challenge lies in determining the effort required to generate a feasible plan. This decision-making process is crucial for efficient planning: If an attempt fails at iteration N, should the agent continue refining the current plan or explore an alternative strategy? Addressing this question requires a structured approach that balances efficiency and robustness. To tackle this problem, we propose a hierarchical planning with low-level pretrained skills.

Our approach leverages Sequencing Task-Agnostic Policies (STAP) (Agia et al. 2023), a trained skill model that provides generalizable action primitives for manipulation tasks. Rather than manually designing task-specific controllers, we use STAP to generate flexible, reusable skills, enabling our system to generalize across different manipulation scenarios. We integrate these pre-trained STAP skills into an infinite completion tree (Toussaint et al. 2024), where applying the pre-trained stochastic policy is used as a computational action. This structure enables efficient exploration of feasible action sequences by dynamically adjusting effort allocation based on task complexity.

However, navigating this infinite search space presents a significant challenge: How can we efficiently guide the search process to maximize task success while minimizing computational overhead? To address this, we introduce a search guidance technique based on estimating the likelihood of success for different action sequences denoted as P. Unlike Q_{value} estimator from STAP, which suffer from poor calibration in continuous domains, P provides more accurate success predictions, improving planning efficiency and execution reliability. By integrating P within our hierarchical framework, we significantly enhance the system's ability to identify promising action sequences and discard infeasible ones early in the search process. Empirical results show that our approach improves success rates up to 33% and reduces planning time by 44% compared to traditional search methods. The combination of STAP p_{score}, infinite completion trees, and our novel success estimator enables more effective decision-making in complex manipulation tasks, making our framework a step toward scalable and generalizable robotic planning.

Related Work

Task and Motion Planning

Task and Motion Planning (TAMP) is a key area in robotics that enables autonomous systems to efficiently execute complex tasks, especially in dynamic and unstructured environments. Unlike traditional motion planning, which focuses on generating collision-free trajectories for individual movements, TAMP combines high-level symbolic task planning with low-level motion generation. This integration allows robots to reason about both discrete and continuous decision

spaces, which is essential for real-world applications where task feasibility depends on both abstract action sequences and the robot's physical constraints and the dynamics of its surroundings.

Recent advancements in TAMP have aimed at improving computational efficiency and scalability. For example, various solvers have been developed to optimize resource allocation while balancing planning accuracy and runtime performance (Garrett et al. 2021). Hierarchical approaches have also been explored to break down the complexity of the problem. One such example is infinite completion trees, presented in (Toussaint et al. 2024), which efficiently schedules computational resources within hierarchical planning processes, demonstrating that structured decomposition can significantly enhance performance in large-scale robotic applications.

Despite these advancements, challenges remain due to the interdependencies between task planning and motion feasibility, the high-dimensional search space, and the uncertainties in real-world robotic applications. To overcome these challenges, researchers are focusing on more sophisticated heuristics, learning-based approaches, and optimization techniques to improve the robustness and generalizability of TAMP frameworks.

Robot Skill Learning

The study of composable manipulation skills is critical for enabling robots to generalize across tasks and adapt to dynamic environments. One of the major challenges in this domain is designing representations that support efficient learning, generalization, and reuse of acquired skills across various scenarios. Learning from Demonstration (LfD) has emerged as a widely adopted paradigm in this context, allowing robots to learn complex behaviors by observing human demonstrations (Ravichandar et al. 2020; Garrett et al. 2021). LfD enables the acquisition of manipulation skills without requiring explicit programming, making it particularly useful for non-expert users in unstructured environments.

A common approach in skill learning involves integrating Dynamic Movement Primitives (DMPs) into neural networks. DMPs model movement trajectories as dynamical systems, providing a flexible and robust representation for smooth, continuous motions. Recent work has enhanced DMP learning capabilities using deep learning techniques to optimize the parameters governing these systems (Ridge et al. 2020; Yang et al. 2018; Agia et al. 2023; Sudry et al. 2023), leading to significant improvements in generalization, and enabling robots to handle a broader range of manipulation tasks.

Building on these advancements, **Sequencing Task-Agnostic Policies (STAP)** (Agia et al. 2023) offer a scalable framework for training manipulation skills and coordinating their geometric dependencies at planning time. This framework allows robots to solve problem which require a novel combinations of skills not encountered during training, thus enhancing generalization. In STAP, each high-level action is mapped to a corresponding skill, and is associated with a learned Q-value, Q, that estimates the likelihood of successful execution from a given configuration c_i. STAP generates all possible high-level plans and selects the one with the highest score, optimizing for task success. The efficiency of this process is enhanced by pruning the search tree, and leveraging prior knowledge about problem length is effective only for problems with short, high-level plans. In such cases, this approach reduces computational complexity and improves scalability.

Search Techniques in Planning

Search trees are a fundamental computational structure in automated planning, used to systematically explore sequences of actions that lead to desired goals or states. The process begins with an initial state (the root of the tree) and iteratively expands by applying possible actions to generate successor states, represented as nodes. This expansion continues until a goal state is reached, or the search space is exhausted. The structure and efficiency of the search tree are critical in determining the feasibility and computational complexity of planning algorithms, especially in large or high-dimensional state spaces.

Classical search algorithms such as depth-first search (DFS) and breadth-first search (BFS) systematically explore the tree. DFS prioritizes deeper exploration, while BFS ensures exhaustive expansion at each depth level. More sophisticated methods, such as heuristic-guided search algorithms (e.g., A*), use domain-specific heuristics to evaluate the expansion of nodes based on estimated cost, goal proximity, or other factors (Hart, Nilsson, and Raphael 1968). These heuristics enhance search efficiency by directing exploration toward promising paths and reducing unnecessary computations.

However, search trees face exponential growth in complex planning problems, which demands effective pruning strategies to eliminate unpromising branches and manage computational resources efficiently. Techniques like alpha-beta pruning in adversarial search, constraint-based pruning, and domain-specific heuristics help mitigate the combinatorial explosion and improve tractability. Additionally, probabilistic and learning-based approaches, such as Monte Carlo Tree Search (MCTS) and deep reinforcement learning, hold promise in optimizing search tree exploration by balancing exploitation and exploration (Silver et al. 2016).

Effort Level Search (ELS) (Toussaint et al. 2024) addresses this issue by utilizing infinite completion trees. An infinite completion tree consists of a root node and a successor function that returns child nodes. Each node in the tree is associated with a state variable representing the total computational effort invested in that node. A node is considered complete when the combinatorial effort exceeds what is necessary, with each node having a latent computational effort required for completion. During each iteration, the algorithm selects an incomplete node and allocates additional computational effort to it. Effort is distributed uniformly across the search space until the goal is reached. However, this approach can lead to inefficient search behavior and wasted effort, as the computational resources are not always directed towards the most promising paths.

Despite significant advancements, challenges remain in

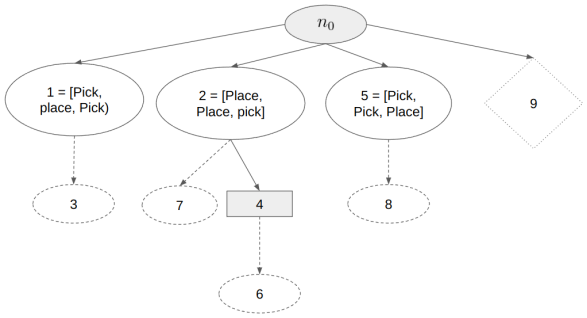

Figure 1: tree search

the scalability, real-time decision-making, and handling uncertainty within search-based planning frameworks. Future research may focus on integrating graph-based representations, sampling-based motion planners (e.g., RRT), and hybrid planning techniques to enhance search efficiency and adaptability in complex, dynamic environments. The intersection of search-based planning with machine learning is also an active area of research, offering potential breakthroughs in autonomous decision-making and real-world robotic applications.

Problem Formulation

We begin by formalizing long-horizon tasks that require the sequential execution of learned skills. A task is defined by an agent interacting with the environment in discrete time steps (Sutton, Barto et al. 1998). At time step t the state is represented by a configuration, $c_t \in C$, with both continuous and discrete elements depending on the problem, E.g. for manipulation tasks c_t could describe the items in the scene with poses, the state of the robot with joint positions and velocities, and discrete elements such as collisions between various items and the robot. We denote the given starting configuration as c_0. We follow the STAP formalism and formulate our problem such that, at each time step, the agent executes a low-level action $u_t \in U$. The environment then transitions **deterministically** to a new state c_{t+1} according to a known dynamics function $f : C \times U \to C$. The goal of the agent is to reach a subset of goal states $C_g \subseteq C$ with the least cost from the initial state c_o.

$$T^* = \min T, \text{s.t } c_{t+1} = f(c_t, u_t), c_T \in C_g \qquad (1)$$

, and the primary challenge is **efficiently** computing a low-level actions sequence, especially when the optimal solution cost T^* is large.

Abstract representation: To more efficiently plan for long trajectories, TAMP uses a higher level of abstraction and maps each low-level state to an abstract high-level state, allowing us to find shorter plans in the high-level representation. To facilitate structured decision-making, we define a discrete high-level abstraction space S, a high-level action space A, and a graph G whose nodes represent states in S. For each state $s \in S$, we define a discrete subset of valid high-level actions $A_s \subseteq A$, where each action in A_s corresponds to an edge in the graph originating from s. We note that actions in A may have (possibly continuous) parameters; in such cases, we merge all parameterizations that lead to the same next state into a single edge to maintain a finite graph. To translate between the high and low-level representations of the problem, we used a mapping function $H : C \to S$ from STAP, which establishes a correspondence between environment states and their high-level representations. We further assume that C_g corresponds to a target high-level goal state $S_g \in S$. We also overload our notation and denote $c \in S$ to denote a configuration for which $s = H(c)$. A high-level action $a \in A_s$, corresponds to an abstract, discrete transformation in the configuration space but necessitates the execution of multiple low-level actions—such as motor control commands—to be realized in the physical robot space.

Note that when traveling on edge (s, s') using a, we always start from a configuration c_s that is connected to c_0, and by executing a, we discover $c_{s'} \in s'$. Finally, the objective viewed in high-level abstraction is to find a path on the high-level graph that connects $s_0 = H(c_0)$ to S_g. A high-level plan, defined as P, is a valid sequence of actions that transforms the initial state into the goal state, ensuring that each action satisfies its preconditions and updates the state with its effects. Each high-level plan is represented as

$$P_i = (s_0, s_1, \ldots, s_{l_i} = s_g)$$

Given a sequence of configurations, every $H(c_t) \neq H(c_{t+1})$ corresponds to an edge traveled in G using the plan.

We follow (Agia et al. 2023), which provides a translation between a high-level action a and its corresponding sequence of low-level actions. We assume the existence of a corresponding skill, denoted as $\psi_a : C \to U$, which outputs low-level actions given a configuration. To execute a transition in the high-level graph starting from $s \in S$ using a, the agent begins in a previously discovered configuration $c \in s$ and sequentially predicts actions with ψ_a until it reaches $c' \notin s$ (i.e., $H(c') = s' \neq s$). The skill ψ_a is considered successful if it moves c_i to c_{i+1}, where $c_i \in s_i$ and $c_{i+1} \in s_{i+1}$, effectively mapping a_i to a transition $s_i \to s_{i+1}$. The skill is deemed to have failed if this condition is not met.

Reflecting back on the objective in Eq. 1, we see that an effective solution would need to balance exploration in the abstract high level graph vs. computing edge traversals by finding specific action sequences.

Example: To clarify the formulation, we present an example based on the robotic manipulation environment used in our experiments. In this setting, a *configuration* refers to a vector encoding the robot's joint positions along with the poses of various colored cubes. The initial configuration, denoted by c_0, includes both the robot arm state and the object poses. The planner has access to STAP-trained skills, specifically ψ_{pick} and ψ_{place}. In this manipulation scenario, the state s is a discrete representation capturing object relations such as object one is on object two. The action space A consists of the STAP skills, Pick and Place, with their associated parameters. k denotes the grasp pose of an object, represented in the object's coordinate frame.

Method

Our objective is to solve the given planning problem. To do so, we formulate an infinite completion tree whose computational actions include (a) choosing a high-level plan, and (b) following a specific pretrained policy. We start by describing our tree search structure, which is partial expansion (Goldenberg et al. 2014; Felner et al. 2012) and what each node represents. We then explain our search guidance heuristics in an infinite branching factor.

Search Tree Formulation

Following the infinite completion tree framework (Toussaint et al. 2024), we define the root node n_0 as the initial configuration c_0 of the problem. Each node has the following attributes:

c_i : configuration

s_i : High-level state, defined as $H(c_i)$.

P_i : A list of all high-level plans we found from the current node state s_i to the goal state s_g.

cnt_i : A counter that tracks the number of attempts made to solve each high-level plan, with an initial value of 0.

At the start of the search, the root node n_0 is initialized with configuration c_0, high-level state $s_0 = H(c_0)$, an empty list of high-level plans $P_0 = []$, and an empty dictionary $cnt_0 = \{\}$. The planner has two computational actions: it can either attempt to solve a low-level plan or generate a new high-level plan for n_0.

Initially, since no high-level plans exist, the planner must generate an initial high-level plan p_0 and add it to P_0, resulting in $P_0 = [p_0]$. We use breadth-first search for our high-level planning, as this is fast enough, but a more efficient high-level planner could be used instead without changing any other details in our algorithm. Once at least one high-level plan is available, the planner must decide how to allocate its efforts—either to try generate a new high-level plan p_i while opening 1,000 nodes or by attempting to find a low-level trajectory implementing one of the high-level actions in an existing high-level plan, see algorithm 1. The process for selecting whether to solve a low-level plan or generate a new high-level plan, as well as determining which low-level state to select for a given high-level state, will be explained in the next section.

The selection process for solving a low-level plan proceeds as follows:

1. Retrieve the set of all high-level plans, set(P), from all nodes.
2. Select a high-level plan p_i.
3. Choose a low-level state c_i from the initial high-level state s_i of the plan.
4. Execute skill ψ_a given low-level state c_i.

Given a plan p_i between s_i and s_n, $p_i = [s_i, s_j, \ldots, s_n]$, if the planner identifies a low-level action corresponding to skill that transitions the system from $c_i \subseteq s_i$ to $c_j \subseteq s_j$ correctly, it generates a new node n_j with configuration c_j, high-level state s_j, high-level plans $p_j = [s_j, \ldots, s_n]$ (which are suffixes of p_i), and an updated counter $cnt_j = \{p_j : 0\}$ (see Figure 1).

Note that in our problem, the search process plays a pivotal role by allowing the system to revisit and resolve previously addressed motion planning challenges. This enables the exploration of alternative configurations c that may facilitate subsequent planning steps. Such a dynamic approach empowers the algorithm to adapt and continuously refine its decision-making over time. For example, the algorithm may reopen a previously explored node that was solved already to find a new configuration, such as node 2, even after processing nodes 4 and 7 (see Figure 1), where the dashed node is not solved, and the solid node is solved. This flexibility highlights the algorithm's capacity to revise plans, in contrast to the STAP approach, which commits prematurely to a high-level plan and cannot reconsider alternatives. In comparison, our method incrementally explores multiple high-level plans, enabling the discovery of improved solutions. A central challenge is balancing between expanding a new high-level plan and refining an existing one. A scoring function governs this decision-making process, the details of which are presented in the following section.

Heuristics

The decision to initiate a new high-level plan or refine an existing one is guided by a scoring function. The score for opening a new high-level plan is computed using the estimated search effort. Sophisticated methods based on deep learning have been proposed (Sudry and Karpas 2022). In our work, we have used a simple counting-based heuristic, combined with a bandit-based approach inspired by (Kocsis and Szepesvári 2006), and is defined as:

$$\text{Score}_{\text{high}} = \frac{1}{L_{\text{nodes}}} \cdot W$$

where L_{nodes} is defined as follows: if no high-level plan has been found, $L_{\text{nodes}} = 1$; otherwise, it corresponds to the number of nodes expended in the most recent plan that was generated in the high-level planner. W is a fixed number that balances high-level and low-level.

The score for selecting and refining an existing motion planning plan is determined using score estimators (see Section *Score Estimator*). Specifically, we employ our proposed P estimator, which defines the score as:

$$\text{Score}_{\text{low}} = \frac{1}{1 + cnt_i(p_j)} \cdot \max(0.1, \text{state}_{\text{score}})$$

where $cnt_i(p_j)$ represents the number of previous attempts of node n_i to execute the plan p_j. $cnt_i(p_j)$ can also be viewed as a branching enumerator, similar to P_w in ELS. The variable $state_{score}$ represents the estimated probability of successfully finding an action, as computed by the score estimator, which will be detailed in the following section. At each time step, the algorithm samples from the set:

$$[\text{Score}_{\text{high}}, \text{Score}_{\text{low(i)}}, \ldots, \text{Score}_{\text{low(j)}}]$$

Where $Score_{low(i)}$ is the score of plan p_i. If solving the low-level plan p_i is selected, there may be several nodes in

the search tree that contain p_i as a high-level plan – either different low-level configurations from the same high-level state, or even from different high-level states for which the same plan applies. We must choose only one low-level configuration in which we apply the skill corresponding to the first high-level action in p_i, so we collect all m nodes that contain p_i in P. Given a list of nodes $[n_i, n_j, \ldots, n_m]$, the planner samples a configuration using the scoring function:

$$\frac{1}{1 + cnt_i(p_j)}$$

Our algorithm differs from ELS, which follows a breadth-first search (BFS) strategy by sequentially evaluating each option. It also diverges from STAP, which prioritizes high-level expansions until reaching a predefined maximum depth, at which point it selects the highest-scoring feasible plan. In contrast, our algorithm is not constrained by a predefined maximum depth, allowing for more flexible and adaptive exploration.

Score Estimator

Accurately estimating the success rate of a behavior policy for a given action $a \in A$ applied at state c is essential for efficiently guiding the search process. This estimation involves evaluating the probability of successfully finding a trajectory that reaches the expected topological state, denoted as $a[\mathrm{H}(c')]$, at each node. In our approach, we compare two primary methods: Q-values from STAP and our estimator, P_{score}. The P_{score} represents the predicted probability of success for a given low-level state-action pair, while Q serves as an alternative method for estimating transition feasibility.

The success prediction module, denoted as P_{score}, is implemented as a neural network that takes as input the current state c and an embedded representation of the high-level action under consideration. The state representation is identical to that used as input to the skill policies, promoting representational consistency and reuse across components. The network outputs a scalar value in the range $[0, 1]$, which corresponds to the estimated probability that the given high-level action will succeed when executed from state c.

To train the score predictor, we collect data by executing random high-level actions in a simulation. Each interaction is annotated with a binary success or failure label, indicating whether the action achieves its intended goal. Notably, this training process is fully decoupled from the training of the skill policies and does not rely on expert demonstrations or finely tuned controllers. Because randomized interactions in simulation can be generated at scale with minimal cost, training the score predictor introduces little additional overhead. Nevertheless, it provides significant utility during planning by enabling the system to prioritize high-level actions that are more likely to succeed.

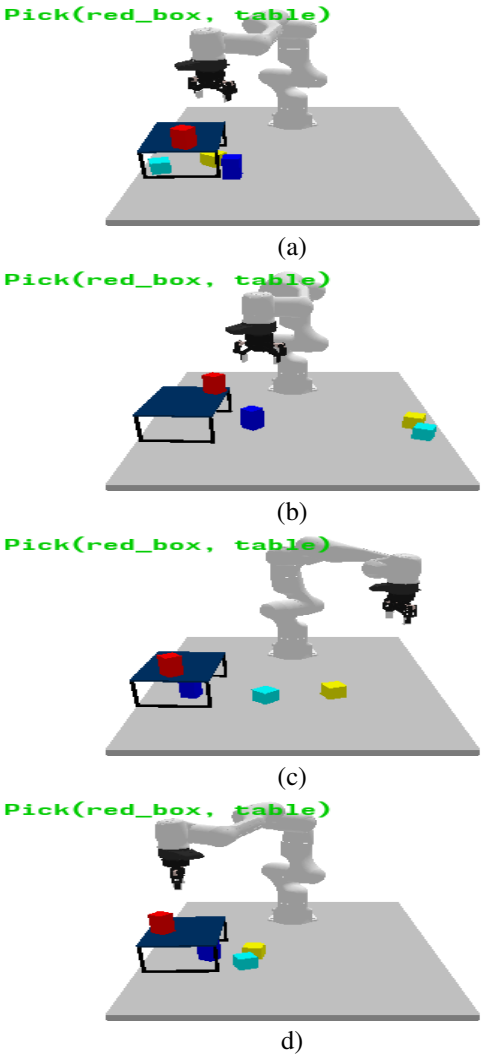

Figure 2: Examples of generated environments

Experiments

We implemented our algorithm[1] on top of the STAP framework (Agia et al. 2023), and performed a set of experiments, designed to investigate the following aspects: (1) a comparison of our algorithm with baseline approaches, (2) an analysis of the search behavior, and (3) an evaluation of the performance of the score estimators, Q and P.

Success Rate of Different Algorithms

To assess the performance of our algorithm, we used the Constrained Packing environment from (Agia et al. 2023) as our method employs a different low-level planner than the trajectory optimizer used in LGP, where the robot is tasked with placing a fixed number of objects onto a rack. To simulate conditions that the robot might encounter in an unstruc-

[1]Our code is available at https://github.com/matansudry/Task-and-Motion-Planning-using-Infinite-Completion-Tree-and-Agnostic-Skills

Algorithm 1: Search algorithm

Input c_{init} Initial configuration state and s_g high-level goal state

Output Low-level plan if found

$s_{init} = H(c_{init})$
while Not timeout **do**
  options = [$OpenNewHighLevelPlanScore()$]
  options.append($GetAllOpenedPlansScores()$) ▷ add all nodes on the open list with their scores
  option = $SelectOption$(options) ▷ sample one option based on scores
  if option == high-level **then**
    $GetHighLevelPlan()$ ▷ add plan to $OPEN$
  else
    PlanFound = $RolloutLowLevelPlan(plan)$ | $plan \in OPEN$ ▷ Using skill ψ_a corresponds to first high-level action in the high-level plan
    if PlanFound **then**
      Return Plan
    end if
  end if
end while

Algorithm 2: RolloutLowLevelPlan(plan)

for i in len(plan) **do**:
  s_i, a_i = plan[i] ▷ (a_i is the action in state s_i of the plan
  $c = ConfigurationRandomSelect(s_i)$
  $\psi_a = GenerateSkill(c, a_i)$
  action succeed, $c_j = ExecuteLowLevelSkill(\psi_a)$
  if action succeed **then**
    $AddNode(c_j)$ ▷ Node c_j added to the tree and can be selected in the next round
  else
    Return False
  end if
end for
Return True

tured, real-world environment, some objects are designated as interfering objects. These objects are initialized in configurations not seen during training, such as being stacked, positioned behind the robot's base, or tipped over. Although the task planner can choose these interfering objects for placement on the rack, the skills have not been trained to handle these configurations. The environment is illustrated in Figure 2. The primitive state is defined as a matrix, where each row represents an object in the scene. The first row corresponds to the object that the robot needs to grasp, the second row represents the object currently being held, and rows three through eight represent the remaining objects in the scene. The skills involved in this task are *Pick* and *Place*, each defined with parameter k, which is a parameterized manipulation primitive, defined as flow;

Pick(obj): the parameter k represents the action grasp pose of obj in the coordinate frame of obj.

Algorithm 3: GetHighLevelPlan()

for i in range(1000) **do**:
  plan = $ExpandNode()$ ▷ open a node in the high-level planner
  if plan **then**
    $AddNode(plan)$ ▷ adding plan to our tree as a node
    return plan
  end if
end for
return False

Place(obj, rec): the parameter k represents the action placement pose of obj in the coordinate frame of rec.

We evaluate our algorithm against the different flavors of ELS (Toussaint et al. 2024) round-robin, $p_w = 3, p_c = 1$, and $p_w = 1, p_c = 1$, and ELS augmented with Q from STAP as an estimator. To assess performance, we designed three levels of problem difficulty: Easy, Medium, and Hard. In the Easy setting, two objects are initially stacked on the table and must be placed as towers on the rack. The Medium setting involves three objects, while the Hard setting consists of four.

Investigating Search Behavior

Table 1 shows the number of problems solved within a time limit of 1,200 seconds and the corresponding solution times. The P_{score} estimator consistently outperformed the other algorithms in solving the majority of the test problems. However, the precise reasons behind this improvement merit further exploration. To better understand the underlying dynamics, we conducted an in-depth analysis of the search behavior within the tree. The results revealed that P_{score} typically opened the fewest number of high-level plans, focusing its search efforts predominantly on low-level nodes. In contrast, ELS tended to open a significantly larger number of high-level plans, as shown in Table 1. Moreover, the analysis highlighted that p_{score} expanded fewer low-level nodes than q_{value} in problems of medium difficulty, suggesting that it was more efficient in these cases. However, in easier problems, p_{score} opened a greater number of low-level nodes compared to q_{value}, though still fewer than ELS. These observations suggest a nuanced search strategy where p_{score} adapts its behavior depending on the difficulty of the problem. In contrast, ELS consistently expanded significantly more nodes across the board. This difference in search behavior provides insight into why ELS, which employs a breadth-first search (BFS) strategy, is less efficient. By indiscriminately expanding the search tree in a breadth-first manner, ELS expends unnecessary computational resources. On the other hand, P_{score} benefits from a more focused search approach, which strategically allocates resources. This targeted search process allows for better optimization and overall efficiency within the search tree, resulting in improved problem-solving performance.

Algorithm	Difficult	Problems	#Solved	Avg time[s]	#high level plans	#low level steps
p_{score}	Easy	55	**45**	**167.7**	1.2	131.7
q_{value}			42	213.4	1.5	110.3
ELS(Round Robin)			42	453.2	7.9	397.0
$ELS(p_w = 3, p_c = 1)$			20	314.3	1.1	67.8
$ELS(p_w = 1, p_c = 1)$			23	351.9	2.2	84.52
p_{score}	Medium	37	**16**	293.2	1.2	286.8
q_{value}			12	389.3	1.4	304.8
ELS(Round Robin)			15	519.6	2.0	444.1
$ELS(p_w = 3, p_c = 1)$			1	**196.84**	1.0	60.0
$ELS(p_w = 1, p_c = 1)$			2	259.86	1.8	106.0
p_{score}	Hard	17	**1**	**593.0**	1	451.0
q_{value}			0	N/A	N/A	N/A
ELS(Round Robin)			0	N/A	N/A	N/A
$ELS(p_w = 3, p_c = 1)$			0	N/A	N/A	N/A
$ELS(p_w = 1, p_c = 1)$			0	N/A	N/A	N/A

Table 1: Comparison between our algorithm denoted as p_{score} and different baselines - ELS Round robin (RR), $p_w = 3$, $p_w = 1$, and q_{value}, which is the same planner with the Q estimator. We compare the number of successes, the time to reach a solution, the number of high-level plans and low-level steps required to solve three difficulties: easy, medium, and hard.

Score Estimator

In our proposed algorithm, we developed the P_{score} estimator to improve the performance of the STAP estimator Q_{value} and enhance the overall effectiveness of the planner. To evaluate and compare the effectiveness of these two estimators, we executed the planner in various scenarios and compared the empirical success rates with their corresponding estimated values. Figure 3 provides a visual comparison between P_{score} and Q_{value} across different states, offering insight into their relative performance. In these figures, the X-axis represents different instances or test cases, while the Y-axis denotes the difference between the absolute error for P_{score} and Q_{value} relative to the ground truth (GT). Specifically, the Y-axis shows the value of abs(GT $- P_{score}$) $-$ abs(GT $- Q_{value}$), where a negative value indicates that P_{score} was closer to the ground truth compared to Q_{value}. These plots reveal that, in terms of proximity to the ground truth, the P_{score} estimator consistently outperformed Q_{value} in a significant proportion of cases. In particular, P_{score} outperformed Q_{value} in 66% of the pick scenarios and 61% of the place scenarios, demonstrating its superior predictive accuracy in these contexts.

Conclusion

In this work, we present a novel hierarchical planning approach that combines hierarchical planning with efficient search strategies. Our method integrates learned skills from the STAP framework with infinite completion trees and a success rate estimator to guide the planning process. The results demonstrate that our approach outperforms existing baselines, underscoring the critical role that accurate success rate estimation plays in efficiently navigating the search space. By leveraging a hierarchical structure, we optimize the search process, enabling the planner to focus resources where they are most needed. This work represents a significant advancement toward a more generalizable hierarchical planning framework. Notably, it marks the first successful integration of infinite completion trees with learned skills, all guided by a success estimation mechanism, paving the way for more flexible and adaptive robotic planning systems.

Future Work

While our hierarchical planning algorithm has shown improvements in efficiency and effectiveness for sequential manipulation tasks, there remain several avenues for further enhancement. A primary focus for future work should be on generalizing our approach beyond the constrained packing problem to a broader range of TAMP challenges. These could include tasks such as deformable object manipulation, multi-agent coordination, and robotic assembly, each of which introduces additional complexities that require more sophisticated planning capabilities. To extend our algorithm's applicability, future research will need to focus on refining the success estimator and advancing skill learning for diverse task distributions, ensuring that the planner can handle the increased variability in these more complex domains.

Another important direction for future research lies in improving the success rate estimator itself. One promising avenue is the incorporation of uncertainty-aware techniques, such as Bayesian neural networks or ensemble methods, to enhance the reliability and confidence of the estimator's predictions. This would provide more robust decision-making in uncertain environments, a key consideration for real-world applications. Furthermore, the development of adaptive search strategies that can dynamically balance exploration and exploitation based on task complexity and execution feedback holds considerable promise. These strategies could potentially leverage reinforcement learning techniques to fine-tune search behaviors in real-time, optimizing the trade-off between exploring new options and exploiting known successful actions.

Advancements in these areas will not only increase the efficiency of hierarchical planning but also improve its adaptability and overall performance in diverse, real-world ma-

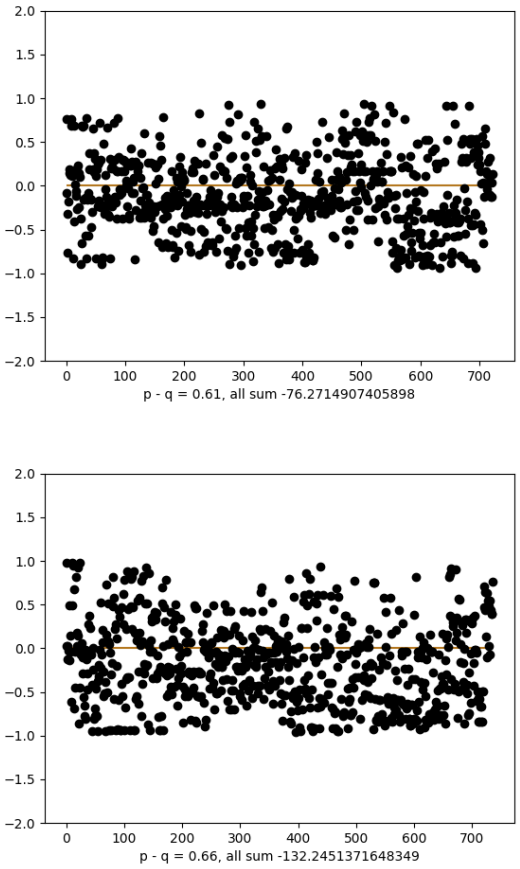

Figure 3: Comparison between P and Q on Place and Pick actions, top is Place. The X-axis represents different instances or test cases, while the Y-axis denotes the difference between the absolute error for P and Q relative to the ground truth (GT)

nipulation tasks. By incorporating these improvements, we believe that hierarchical planning approaches can become an even more powerful tool in the realm of robotic manipulation, enabling systems to tackle increasingly complex and dynamic environments.

Acknowledgements

This work was supported by a grant from the Israeli Planning and Budgeting Committee.

References

Agia, C.; Migimatsu, T.; Wu, J.; and Bohg, J. 2023. Stap: Sequencing task-agnostic policies. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 7951–7958. IEEE.

Felner, A.; Goldenberg, M.; Sharon, G.; Stern, R.; Beja, T.; Sturtevant, N.; Schaeffer, J.; and Holte, R. 2012. Partial-expansion A* with selective node generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, 471–477.

Garrett, C. R.; Chitnis, R.; Holladay, R.; Kim, B.; Silver, T.; Kaelbling, L. P.; and Lozano-Pérez, T. 2021. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1): 265–293.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R. C.; and Schaeffer, J. 2014. Enhanced partial expansion A. *Journal of Artificial Intelligence Research*, 50: 141–187.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Kocsis, L.; and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, 282–293. Springer.

LaValle, S. M. 2006. *Planning algorithms*. Cambridge university press.

Ravichandar, H.; Polydoros, A. S.; Chernova, S.; and Billard, A. 2020. Recent advances in robot learning from demonstration. *Annual review of control, robotics, and autonomous systems*, 3(1): 297–330.

Ridge, B.; Gams, A.; Morimoto, J.; Ude, A.; et al. 2020. Training of deep neural networks for the generation of dynamic movement primitives. *Neural Networks*, 127: 121–131.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484–489.

Sudry, M.; Jurgenson, T.; Tamar, A.; and Karpas, E. 2023. Hierarchical planning for rope manipulation using knot theory and a learned inverse model. In *Conference on Robot Learning*, 1596–1609. PMLR.

Sudry, M.; and Karpas, E. 2022. Learning to estimate search progress using sequence of states. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, 362–370.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Toussaint, M. 2015. Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In *IJCAI*, 1930–1936.

Toussaint, M.; Ortiz-Haro, J.; Hartmann, V. N.; Karpas, E.; and Hönig, W. 2024. Effort Level Search in Infinite Completion Trees with Application to Task-and-Motion Planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 14902–14908. IEEE.

Yang, C.; Chen, C.; He, W.; Cui, R.; and Li, Z. 2018. Robot learning system based on adaptive neural control and dynamic movement primitives. *IEEE transactions on neural networks and learning systems*, 30(3): 777–787.