

Learning Problem Decomposition for Efficient Sequential Multi-object Manipulation Planning

Yan Zhang, Teng Xue, Amirreza Razmjoo, Sylvain Calinon

Abstract—We present an efficient task and motion replanning approach for sequential multi-object manipulation in dynamic environments. Conventional Task And Motion Planning (TAMP) solvers experience an exponential increase in planning time as the planning horizon and number of objects grow, limiting their applicability in real-world scenarios. To address this, we propose learning problem decompositions from demonstrations to accelerate TAMP solvers. Our approach consists of three key components: *goal decomposition learning*, *computational distance learning*, and *object reduction*. Goal decomposition identifies the *necessary* sequences of states that the system must pass through before reaching the final goal, treating them as subgoal sequences. Computational distance learning predicts the computational complexity between two states, enabling the system to identify the temporally closest subgoal from a disturbed state. Object reduction minimizes the set of active objects considered during replanning, further improving efficiency. We evaluate our approach on three benchmarks, demonstrating its effectiveness in improving replanning efficiency for sequential multi-object manipulation tasks in dynamic environments. Accompanying website: <https://sites.google.com/view/problem-decomposition-ral>

I. INTRODUCTION

Numerous daily activities, from cooking [1] to furniture assembly [2], require sequential manipulation planning over multiple objects. These tasks often involve planning over various interactions among robots and objects under spatial-temporal constraints, and over multiple steps with sparse rewards [3]–[5], posing significant challenges for autonomous robotic solutions. Task and Motion Planning (TAMP) has emerged as a powerful framework for tackling complex sequential multi-object manipulation tasks [1, 3]. TAMP integrates task planning with motion planning by combinatorially searching sequences of abstracted actions and corresponding motion parameters [6]–[8]. Consequently, as the planning horizon and the number of environmental objects grow, TAMP solvers require exponentially longer planning time [1, 9, 10]. This time-intensive planning thus poses challenges when using TAMP solvers in closed-loop systems requiring to efficiently react to disturbances in real-world scenarios.

A considerable number of manipulation tasks typically require passing through several subgoals in sequence to achieve the final task goal. For example, cooking a meal often involves preparing various ingredients in a specific order as dictated by a recipe. On one hand, finding solutions that follow these subgoal sequences in the correct order requires identifying a small subset of solutions from a vast solution space, which significantly increases planning time. On the other hand, we argue that these subgoal sequences can be

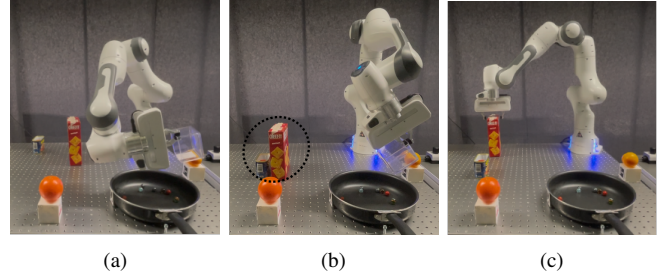


Fig. 1: The Franka Emika robot arm responds robustly to disturbances (black dashed line) in the positions of the crackers box and meat can, using our efficient task and motion replanning approach, which is enabled by integrating learned problem decomposition from demonstrations.

extracted from demonstrations and used to decompose complex tasks into simpler subtasks. For instance, determining the correct order for processing ingredients without a recipe would involve extensive trial-and-error planning. However, based on prior experience preparing the same meal, one can summarize a recipe and focus only on reaching the next closest subgoal in that sequence—thereby simplifying the planning process. More importantly, these summarized subgoal sequences exhibit strong robustness to variations in state and environment. For example, a meal can typically be prepared by following the same recipe, regardless of the initial placement of ingredients, the layout of the kitchen, or the shapes of objects. This intuition supports our argument that learning subgoal sequences not only enables goal decomposition but also imparts robustness to task variations. We further argue that this resulting decomposition and robustness can greatly facilitate the design of efficient TAMP solvers for sequential multi-object manipulation planning under disturbances.

We proposed to enhance classical TAMP solvers with problem decompositions learned from demonstrations. The proposed problem decomposition consists of three key components: *goal decomposition*, which analyzes the collected demonstrations to identify the *necessary* sequences of states the system must pass through before reaching the final goal, treating them as subgoal sequences; *computational distance learning*, which uses the same set of demonstrations to train a Graph Neural Network (GNN) that predicts the importance of environmental objects and uses the number of important objects as a computational distance metric to select the closest subgoal from a disturbed state; and *object reduction*, which runs multiple TAMP solvers with different object sets to find a feasible plan that connects the disturbed state to the predicted closest subgoal, improving replanning efficiency.

In summary, the main contributions of this paper are:

- 1) We propose a novel subgoal formulation and generation approach, where partial state sequences that must be sequentially traversed are considered as subgoal sequences for goal decomposition, and we generate them from demonstrations by casting this as a sequential pattern mining problem;
- 2) We introduce a computational distance learning method that approximates computational distance between states via object importance prediction, facilitating the selection of the closest subgoal for efficient task and motion replanning;
- 3) We develop an object reduction mechanism that preserves solution completeness while accelerating replanning efficiency;
- 4) Integrating above three learned problem decomposition modules with classical TAMP solvers, we introduce an efficient task and motion replanning framework for sequential multi-object manipulation planning in dynamic environments.

II. RELATED WORK

A. Learning Subgoals for Long-horizon Planning

Our motivation of learning subgoal sequences from demonstrations is based on previous research works that indicate subgoals can be used to decompose a long-horizon planning problem into simpler subproblems, thus accelerating long-horizon planning efficiency [11, 12]. However, these works assume that subgoals are included in the training dataset and train a neural network to predict online the next subgoals for accelerating task planners. Similarly, with internet-level dataset, Large Language Models (LLMs) can be used to infer subgoals to accelerate the learning efficiency for solving complex sequential manipulation tasks [4, 5]. Our work is complementary to such works in the sense that we assume no access to such subgoals in the dataset and aim to generate such subgoals from unlabeled dataset in an unsupervised way. Recently, *Elimelech et al.* [13] identifies critical states from past experiences and use segmented state trajectories as higher-level action abstractions for speeding up task planning [14]. Our approach mainly differs from [13, 14] in the definition and usage of subgoals. In their methods, subgoals are defined as critical states that maximize abstraction level and minimize state trace segmentation. The traces between consecutive subgoals are then used as higher-level symbolic actions to decompose tasks. In contrast, we identify *partial* states that must be traversed to reach the task goal as subgoals, and use them as checkpoints during replanning—without reusing the state traces between them—as such traces may need to be replanned in integrated TAMP. *Levit et al.* [15] suggested identifying subgoals in the bottleneck region of the environment to improve the resolution of 2D puzzle problems with optimization-based TAMP solvers. In their work, subgoals are generated based on environment geometry to alleviate local optima issues in optimization-based motion planning. Differently, we define must-pass states as subgoals and use them to shorten the planning horizon and enhance the planning efficiency of TAMP solvers.

B. Task and Motion Planning

Our work falls within the domain of learning for Task and Motion Planning (TAMP), where machine learning methods are applied to leverage previous experiences to improve the planning efficiency of TAMP solvers. Existing works include learning value functions as cost-to-go heuristics [16], learning action feasibility [1, 17, 18] to prune infeasible and computationally expensive motion-planning validations, and identifying important objects [9] or their associated streams [19], all of which significantly speed up TAMP solvers. Our approach differs from these methods by proposing to learn necessary subgoal sequences as guidance and a computational distance metric for selecting the temporally closest subgoal from a disturbed state, thereby minimizing the planning horizon and number of objects for TAMP solvers, thus enhancing their planning efficiency.

To achieve fast replanning, *Migimatsu et al.* [20] introduced an object-centric TAMP approach, where motion trajectories are executed in the Cartesian coordinates of the corresponding objects, demonstrating reactivity to motion-level disturbances. To address both task- and motion-level disturbances, [21, 22] propose an offline construction of a set of prioritized action plans, which can then be dynamically switched online for fast replanning. The success of this pipeline relies on the completeness of the constructed plan library and cannot effectively handle disturbed states that fall outside the predefined plans. *Xue et al.* [23] explore goal backpropagation to improve the planning efficiency of TAMP solvers and formulate their approach in a closed loop to handle various disturbances. Their method relies on the effectiveness of the back-propagation heuristic, which can be difficult to generalize across different domains.

In terms of accelerating TAMP solvers via planning horizon reduction, *Castaman et al.* [24, 25] propose Receding-Horizon TAMP (RH-TAMP) approaches that iteratively solve a reduced problem over a fixed receding horizon. However, fixed receding horizons often fail to capture the full-horizon spatial-temporal constraints, which can result in increased infeasibility of action skeletons. The work most closely related to ours is Logic Learning from Demonstrations (LogicLfD) [26], which reduces the planning problem to a partial problem by running a TAMP solver to reach the demonstrated trajectory as fast as possible. In contrast, the method presented in this paper represents demonstrations as sequences of necessary subgoals, which enables to accelerate TAMP solvers by reducing both the planning horizon and the number of objects considered. This results in superior reactivity in manipulation tasks with many objects and long planning horizons.

III. METHOD

Figure 2 presents an overview of our efficient task and motion replanning approach, which integrates problem decomposition with TAMP solvers (e.g., PDDLStream [8], used in this paper) for efficient sequential multi-object manipulation planning. Our problem decomposition framework consists of three components: *goal decomposition learning*,

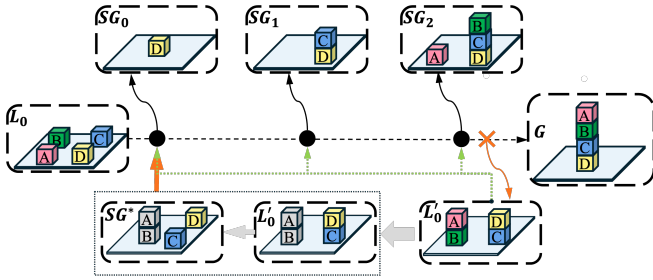


Fig. 2: An overview of our efficient planner, which integrates *goal decomposition*, *computational distance*, and *object reduction* with classical TAMP solvers to enable efficient replanning under online disturbances (orange 'x' symbol). Goal decomposition identifies the *necessary* subgoal sequences ($SG_0 \rightarrow SG_1 \rightarrow SG_2$) offline from demonstrations. The computational distance metric is trained offline using the same set of demonstrations and is used to measure the distance (green arrows) from a disturbed state L'_0 to the subgoals (black circles), enabling the identification of the closest subgoal $SG^* = SG_0$. Object reduction (area with dashed line) considered grey cubes as fixed obstacles while replanning from L'_0 to SG^* so that only necessary objects are involved in. The whole process is repeated iteratively until the final task goal is achieved.

computational distance learning, and *object reduction*. We describe each of these components in detail below. As an extension of Figure 2, we provide a pseudocode description of our efficient task and motion replanning approach on the project website to offer a more detailed explanation of our algorithms.

A. Demonstration Generation

The left image in Figure 3 shows the configuration of objects at a specific time step t . Boolean symbolic variables $s_t \in \mathcal{S}$ are used to define the symbolic state s_t of the environment at time step t . These symbolic variables capture internal properties, geometric information, and spatial relationships between objects—for example, (`clear B`), (`atPose B p1`), and (`on B C`) describe whether object B is clear, its pose in the environment, and whether it is on top of another object C, respectively. A symbolic state s_t is an itemset of Boolean symbolic variables drawn from the set \mathcal{S} , representing symbolic description of all environment objects at time step t .

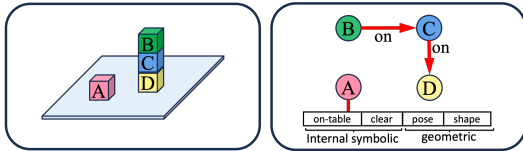


Fig. 3: Illustration of objects' configurations at specific time step and its graph representation.

We assume access to a set of demonstrations that successfully achieve the task goal G across N distinct initial states. In this paper, we use PDDLStream [8] to collect $N = 40$ demonstrations for all tasks in Section IV. The different initial states are generated by randomizing the logical states of environmental objects and their geometric poses. However, these demonstrations could also be obtained from human demonstrations, internet datasets, or past planning experiences of any planner, as long as they include the geometric

states of environment objects (e.g., poses, shapes) so that they can be translated into symbolic states using a predefined state abstraction. We represent the unified dataset of demonstrations as $\mathcal{P} = \{\mathcal{P}^s, \mathcal{P}^g\}$, where \mathcal{P}^s and \mathcal{P}^g denote symbolic and geometric state trajectories, respectively. The i th symbolic trajectory \mathcal{P}_i^s is a sequence of symbolic states $s_{t=1, \dots, k_i}$ of length k_i , while the corresponding geometric trajectory \mathcal{P}_i^g captures the pose trajectories of all objects over time.

B. Goal Decomposition Learning

With the demonstrations, our goal is to identify sequences of *partial states* in \mathcal{P}^s that are consistently traversed to reach the target task goal—regardless of the initial state or the specific action plan chosen. Here, *partial states* refer to symbolic states of a subset of environmental objects in the planning domain. For instance, in SG_0 of Figure 2, only (`ontable, D`) and (`clear, D`) are considered as *partial states*. Although our symbolic states also include geometrical information with geometrical predicates, these symbolic states related to cube D, such as (`atpose, D, p1`), and those indicating relationship between cube D and other cubes are excluded from subgoal SG_0 because they are not essential for achieving the final goal G . Since these partial states must be visited in a specific order to reach the goal, we expect to observe the same sequences of partial states across all demonstrations. We therefore formulate the goal decomposition problem as follows: given a set of demonstrations, identify sequences of partial states that are consistently observed across all demonstrations as subgoal sequences.

Given that the symbolic state s_t is represented as an itemset of Boolean symbolic variables s , the symbolic trajectory \mathcal{P}^s can be expressed as N sequences of itemsets $\mathcal{H} = \{s_{t=1, \dots, k_i}\}_{i=1, \dots, N}$. We then identify the sequences of itemsets that are common across all symbolic trajectories by formulating it as a sequential pattern mining (SPM) problem [27]. In this paper, we adopt PrefixSpan [28], a widely used SPM algorithm known for its simplicity and flexibility in incorporating constraints [27]. For more details on the algorithm, refer to [27, 28].

PrefixSpan operates on the sequence database \mathcal{H} by recursively projecting only those parts of the database that are relevant to a given prefix pattern. This thus greatly reduces the search space and efficiently identifies all the common sequences of itemsets $\{\tilde{s}_{t_m}\}_{m=1, M}$ that meet or exceed a minimum occurrence frequency ratio, defined as *min-support*. Here, t_m is a list of time index, and each \tilde{s}_{t_m} denotes a sequence *partial states* of symbolic states s_{t_m} at key time indices in t_m . M is the total number of discovered sequential patterns. We set the minimum support threshold (*min-support*) to 0.9, targeting those sequential itemsets that appear in at least 90% of the demonstrations.

Using PrefixSpan, we generate a set of itemset sequences including both the target subgoal sequences $SG = \{SG_0, \dots, SG_Z\}$ and all its subsequences, where Z is an unknown variable representing the length of the target

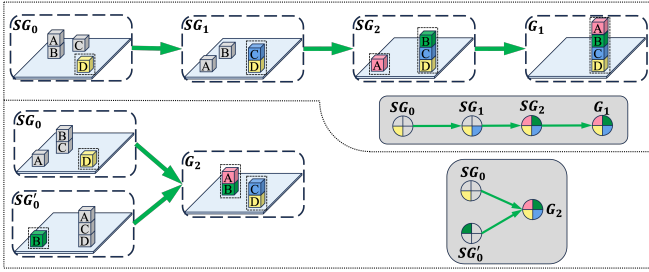


Fig. 4: Illustration of subgoal sequences for two task goals G_1, G_2 in tower construction toy example.

sequence. We iteratively compare the generated sequential patterns to filter out all the subsequences. Correspondingly, we will generate one or multiple sequences of subgoals based on the decomposability of the task goal. Figure 4 illustrated subgoal sequences of two different task goals. The top sequence shows the subgoal sequence for task goal G_1 where blocks should be stacked in a linear sequential order, from SG_0 to G_1 . In this sequence, each subgoal (partial symbolic states of colorful blocks within dashed line) captures a partial state of the environment, with only the colored blocks being relevant for the subgoal, while grey blocks can be at any configuration. Areas with grey background show the corresponding Directed Acyclic Graph (DAG) representation of the sequential structure of the subgoals for G_1 ; In contrast, the lower sequence for G_2 illustrates a multi-modal sequential solution. We can only stack A/C on B/D once B/D are clear; However, to achieve G_2 , it does not matter whether D becomes clear before B becomes clear, or even after A is stacked on B, as the only constraint is that D must be clear before stacking C on it. Consequently, we will observe state trajectories where D becomes clear before or after B becomes clear in demonstrations. PrefixSpan thus identifies that different subgoal sequences (SG_0 and SG'_0) can lead to the same final goal G_2 . This shows that in the same planning domain, different task goals have different numbers of subgoal sequences.

C. Computational Distance Learning

1) *Graph-based Demonstration Representation*: We represent the symbolic and geometric states of demonstrations \mathcal{P} at each time step using a graph structure. In this graph, each object is represented as a node. Symbolic states that describe relationships between objects (e.g., (onblock, B, C)) are encoded as edge features. Other symbolic states that describe intrinsic properties of individual objects (such as (clear, A)), along with geometric information (e.g., pose, shape), are incorporated as node features. The right image in Figure 3 show an example of the constructed graph structure.

2) *GNN-based Computational Distance Learning*: Computational distance refers to the computational time required to find a feasible solution from a disturbed state to a subgoal or the task goal [29]. The exact computational time cost depends on several factors, such as the configurations of the environment at the initial and goal states, the state and action abstraction in TAMP solvers, the corresponding

streams [8] or motion planners, and hardware. These factors make it infeasible to design a general computational distance function that predicts computational time precisely across different problems. Therefore, we propose to approximate the relative computational distance¹ $d(s_1, SG)$ with the number of the important objects required for transitioning from the disturbed state s_1 to a subgoal SG . The motivation behind this representation is that planning time typically increases exponentially with the number of objects involved and the overall planning horizon. Generally, the more objects are involved, the longer the planning sequence is, or the more motion parameters must be determined during the motion planning phase. Thus, we posit that the number of important objects can serve as a useful indicator of relative computational distance.

To learn this relative computational distance metric, we segment demonstrations \mathcal{P} based on the generated subgoal sequences SG . Each segment thus describes the transition process from one subgoal SG_i to the next or a later subgoal SG_{i+1} or SG_{i+z} , where $z \in [1, Z - i]$. In each segment, we identify important objects \mathcal{O}^* for each subgoal transition process by analyzing changes in object states: objects whose states change during the transition are considered important. We then construct a graph-based dataset, where each input consists of the graph representations of the initial and goal states of a segment. The output is an importance score for each object: objects in \mathcal{O}^* are assigned a score of 1, while others receive a score of 0.

We train a GNN model on this dataset to predict object importance. During online execution, when a disturbance is detected, we compute the computational distance $d(s_1, SG_i)$ from the current disturbed state s_1 to each subgoal SG_i based on the number of predicted important objects (i.e., those with scores greater than 0.9). The subgoal with the minimal predicted distance and earliest position in the subgoal sequence is considered the temporally closest subgoal SG^* .

This relational representation is similar to the one presented by Silver *et al.* [9] for predicating the importance of environmental objects in task planning. However, our approach differs in two key ways: (1) we incorporate the geometry properties of objects also as part of the node properties so that the trained GNN model aligns with TAMP solvers; (2) we employ the GNN model not only to approximate the corresponding computational distance but also to predict important objects for object reduction later. Correspondingly, the advantages of our approach to formulate computational distance are twofold. First, by using the same trained GNN model, we can simultaneously predict both the computational distance for choosing the closest subgoals and use the predicted important objects for *object reduction* while replanning to the predicted closest subgoals. Second, since we prioritize reaching the closest subgoal with the fewest important objects, the prediction accuracy

¹We define the actual system state which is different from the expected state after executing the target action as a disturbed state. The system starts to conduct task and motion replanning once detected a disturbed state.

for subsequent subgoals—those requiring prediction over longer horizons—becomes less critical. This results in a more simplified graph machine learning problem and alleviate the issue when requiring GNN to capture global long-horizon dependency among nodes [30].

D. Object Reduction

In this section, we introduce an object reduction strategy to augment TAMP solvers, aimed at improving replanning efficiency when replanning to reach the predicted closest subgoal SG^* . Traditionally, a TAMP solver seeks a feasible plan τ for the problem $P(s_1, SG, \mathcal{O})$, where the planner considers the full set of environmental objects \mathcal{O} to connect the disturbed state s_1 to the target subgoal SG . However, many objects in the environment may be irrelevant to achieving the subgoal, and reasoning over all of them introduces unnecessary computational overhead.

To enable object reduction, we leverage the predicted importance scores from the trained GNN model to dynamically construct reduced object sets for replanning. We begin with an initial importance threshold $\epsilon = 0.9$ and generate a descending sequence of thresholds $\{\epsilon, \epsilon^2, \dots, \min(\epsilon^i, 0.01), 0\}$ with 6 elements. For each threshold, we select a subset of objects \mathcal{O}_i^* whose predicted importance scores exceed the threshold, and treat all remaining objects as fixed obstacles. This results in a set of subproblems $\{P_i\}_{i=1, \dots, 6}$ with progressively larger object sets $\{\mathcal{O}_i^*\}_{i=1, \dots, 6}$ but identical initial and goal states s_1, SG^* . We perform parallel replanning for each subproblem. If any of the subproblems returns a feasible plan, all other processes are terminated. This ensures that we prioritize solving the most efficient, object-reduced problem first, while still preserving the ability to solve the full problem if necessary.

Incorrect predictions from the GNN may exclude critical objects, potentially rendering a subgoal unreachable even if it is otherwise feasible. Moreover, in motion planning, infeasibility due to missing objects can be difficult to detect and may lead to unbounded planning time. To mitigate this, we include a minimal threshold of zero in the threshold list. This guarantees that, in the worst case, all objects are included in the final attempt, ensuring the feasibility of reaching SG^* remains unchanged while significantly improving planning efficiency when accurate importance predictions are available.

IV. EXPERIMENT

A. Benchmarks

1) *B1-Tower Construction*: In this benchmark, the robot is tasked with stacking a set of blocks into a specific configuration, starting from a random initial layout. The robot must compute a feasible task and motion plan using the available primitive actions: *pick*, *place*, *stack*, and *unstack*.

2) *B2-Tool Use*: This task extends the *B1* scenario by introducing a tool that enables the robot to *pull* objects located outside its immediate workspace. The robot must autonomously decide when and how to use the tool, integrating this reasoning into its planning process. In addition

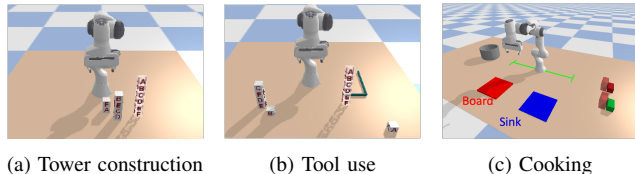


Fig. 5: In (a) and (b), transparent blocks indicate a representative task goal. In the Cooking domain, the Franka manipulator is equipped with a virtual mobile base, enabling movement along the Y-axis within the boundary indicated by green lines.

to the stacking objective, the robot must satisfy inverse kinematics constraints and avoid collisions between the tool and surrounding objects.

3) *B3-Cooking*: In this benchmark, a Franka Emika robot arm mounted on a mobile base (with movement allowed along the Y-axis) is tasked with preparing a meal by sequentially washing, cutting, and cooking ingredients—each represented by colored blocks. The environment includes clutter in the form of transparent red blocks, simulating obstacles in a typical kitchen, which introduces additional collision constraints and limits accessibility to certain ingredients. The robot must also obey temporal constraints, such as processing multiple ingredients in a predefined order.

Available actions include: *pick*, *place*, *wash*, *cut*, *cook*, and *cook-after*, where *cook-after* enforces specific temporal dependencies across different ingredients.

B. Planning Complexity & Decomposability of Task Goals

In this section, we highlight the importance of goal decomposition by analyzing the relationship between a task goal’s decomposability and its planning complexity. We constructed a dataset of up to 40 demonstrations using PDDLStream [8] for two benchmark tasks: *B1 – Tower Construction* and *B3 – Cooking*. For each benchmark, we consider three distinct task goals, each exhibiting varying levels of complexity in terms of subgoal sequences. For example, in benchmark *B1*, the task goals are defined as follows:

- *Goal 0* requires placing all blocks on the table, therefore has no intermediate subgoals.
- *Goal 1* involves forming two distinct stacks of blocks in a specific order, such as *(on A/C, B/D)*, *(on-table B/D)*, therefore has two subgoal sequences.
- *Goal 2* requires building a single stack in a specific order, such as *(on A/B/C, B/C/D)*, *(on-table D)*, thus must be achieved following one subgoal sequence.

We define planning complexity as the average time required to compute a feasible plan for a given task goal, and decomposability as the number of subgoals that can be identified to achieve that goal. We compare the average planning time, average planning horizon (i.e., the number of actions), and the number of subgoals by running PDDLStream across three different task goals and numbers of objects. A summary of these comparisons is presented in Table I.

Our results show that, with the same number of objects and similar planning horizons, planning time increases as the number of subgoals rises. This suggests that goals with more subgoals exhibit higher planning complexity—yet

(a) Planning complexity vs. decomposability in *Tower Construction*

	4 Blocks			6 Blocks			8 Blocks		
	Goal 0	Goal 1	Goal 2	Goal 0	Goal 1	Goal 2	Goal 0	Goal 1	Goal 2
Time [s]	0.2 ± 0.0	0.4 ± 0.0	2.9 ± 0.4	0.3 ± 0.02	3.2 ± 1.3	6.2 ± 0.8	0.5 ± 0.0	179.1 ± 0.8	≥ 180
Horizon	6 ± 0	7.6 ± 1.8	11.6 ± 1.7	10 ± 0	19.7 ± 2.8	18.6 ± 2.2	14 ± 0	24.5 ± 1.7	29.8 ± 3.2
Subgoals	0	2	4	0	4	6	0	6	8

(b) Planning complexity vs. decomposability in *Cooking*

	2 Ingredients			3 Ingredients			4 Ingredients		
	Goal 0	Goal 1	Goal 2	Goal 0	Goal 1	Goal 2	Goal 0	Goal 1	Goal 2
Time [s]	3.4 ± 0.3	3.1 ± 0.1	3.1 ± 0.1	14.5 ± 1.8	26.5 ± 14.5	25.7 ± 13.5	≥ 180	≥ 180	≥ 180
Horizon	10.6 ± 0.7	9.4 ± 1.3	9.4 ± 1.3	14.7 ± 1.0	15.2 ± 0.9	14.3 ± 1.6	20.0 ± 0.6	20.6 ± 1.4	21.8 ± 2.6
Subgoals	3	3	3	3	4	5	4	6	7

TABLE I: Planning complexity vs. decomposability in *Tower Construction* and *Cooking* benchmarks. The 'N/A' symbol indicates tasks where the solution time exceeded 180 seconds, reaching a time-out. Generally, task goals with a higher number of subgoals lead to increased planning horizon and time, but they also exhibit greater decomposability, showing the significance of goal decomposition.

also greater decomposability. For example, in Table Ib, the planning time for reaching *Goal 1* and *Goal 2* increases substantially with the number of subgoals, when cooking a meal with three ingredients. Moreover, while cooking with four ingredients takes over 180 seconds for all task goals, the most complex task goal (*Goal 2*) shows a lower success rate in our evaluations, reflecting the higher average planning time required.

More critically, tasks with the same number of objects but more complex goals generally demand a longer planning horizon and time. As shown in Table Ia, *Goal 2* consistently requires more planning time and a longer horizon across different object counts compared to *Goal 1*, and considerably more than *Goal 0*. However, the number of subgoals also increases correspondingly with the increase of the planning horizon and time, indicating higher decomposability for these complex task goals. Interestingly, although placing eight blocks on the table (*Goal 0* with 8 *Blocks*) is not decomposable and requires relative long planning steps (14) due to the large number of objects, it requires significantly less planning time compared to the other two more complex task goals when involving the same number of objects. This shows that tasks with subgoals are often harder to solve, thus demonstrating the necessity of our proposed goal decomposition.

In conclusion, the proposed goal decomposition method is particularly effective at breaking down complex task goals that require passing through multiple critical states in specific sequential order, which would otherwise lead to long planning time. The more subgoals a task goal has, the more complex it becomes, requiring longer planning time. However, this also increases its decomposability, as each state can serve as a checkpoint, dividing the task into smaller, more manageable subtasks.

C. Replanning Efficiency of Our Approach

This section evaluates the reactivity of our proposed replanning method by comparing it against several baselines: *PDDLStream*, *LogicLfd* [26], *LogicLfd⁺*, and our method without object reduction (denoted as Ours No OR). *LogicLfd* takes a single demonstration and finds the temporally closest seen symbolic state and the plan to reach this state simultaneously by running the *PDDLStream* solver to reach all

symbolic states on the demonstrated trajectory concurrently, then considers the one connected with minimal time as the closest state. *LogicLfd⁺* is a variant that replaces the original demonstration with our discovered subgoal sequences, thereby incorporating goal decomposition and a model-based computational distance, but without any object reduction. To isolate the effect of each module, we also evaluate our full method with goal decomposition and computational distance estimation but without object reduction. This variant helps highlight the efficiency gains contributed by the object reduction mechanism. The following types of online disturbances are considered:

- *L1 Disturbance*: Changes in object configurations that require replanning and subgoal update.
- *L2 Disturbance*: Addition of environmental objects that can be ignored during replanning.
- *L3 Disturbance*: Introduction of additional objects that requires consideration during replanning.

For evaluation, we randomly initialize the environment and introduce disturbances before completing the target task. It is important to note that recovering from disturbances may require not only replanning to the closest subgoal but also to subsequent subgoals. To ensure a fair comparison, we measure the total replanning time from the disturbed state to the final task goal. The tasks conducted in the *Tower Construction* and *Tool Use* domains involves eight blocks (denoted as *Obj8* in the table) under *L1* disturbances and three additional blocks under *L2* and *L3* disturbances, targeting task *Goal 2*. For the *Cooking* tasks, we used four ingredients (denoted as *Obj4*) under *L1* disturbances, with three additional blockers introduced for *L2* and *L3* disturbances. The results are summarized in Table II.

Overall, we observed that vanilla *PDDLStream* struggles to efficiently handle disturbances, achieving a success rate of only 60% under *L1* disturbances in the *Cooking* domain and failing all tasks in *Tower Construction* and *Tool Use* within the 180-second timeout. *LogicLfd* improves upon *PDDLStream* by considering symbolic states on the demonstrated trajectory as subgoals. However, its replanning efficiency is heavily dependent on the quality of the demonstrated trajectory, leading to instability. For instance, in the *Cooking* domain, a single demonstration proves insufficient for successful replanning within the timeout. In contrast, our

	Tower Construction-Obj8			Tool Use-Obj8			Cooking-Obj4		
	L1	L2	L3	L1	L2	L3	L1	L2	L3
PDDLStream	N/A	N/A	N/A	N/A	N/A	N/A	108.8 ± 27.6	N/A	N/A
LogicLfd	6.4 ± 0.6	73.7 ± 43.5	115.8 ± 29.1	18.4 ± 0.6	62.5 ± 24.2	58.0 ± 23.0	165.8 ± 5.5	N/A	N/A
LogicLfd+	4.5 ± 0.4	96.9 ± 53.0	127.3 ± 79.6	20.1 ± 4.4	76.2 ± 50.2	74.2 ± 46.0	38.7 ± 6.0	99.3 ± 26.6	99.5 ± 26.7
Ours No OR	4.3 ± 0.5	46.6 ± 38.8	62.7 ± 55.5	22.8 ± 6.3	79.3 ± 23.3	58.0 ± 23.0	60.0 ± 24.4	118.1 ± 10.4	116.0 ± 4.7
Ours	2.4 ± 0.6	7.5 ± 0.9	7.1 ± 0.8	7.2 ± 0.9	11.7 ± 1.0	12.7 ± 0.6	24.3 ± 4.6	34.1 ± 7.5	34.5 ± 7.8

TABLE II: Comparison of total replanning time [s] from a randomly disturbed state to the task goal across three benchmarks. 'N/A' indicates no successful trials within the 180-second timeout. Results demonstrate that *Ours* achieves significantly better reactivity and efficiency under various disturbances compared to the baseline methods because of the integration of problem decomposition.

generated subgoal sequences guide the TAMP solver toward only the *necessary* subgoals, making *LogicLfd+* more consistent and reliable than *LogicLfd*. Since our learned computational distance is an approximation, it may occasionally select the second closest subgoal, resulting in slightly longer replanning time than a model-based computational distance definition in *LogicLfd+*. However, this model-based approach requires reasoning over all objects, limiting efficiency. In our full method *Ours*, we leverage the approximate computational distance to identify the closest subgoal, then apply object reduction to significantly improve replanning efficiency. As demonstrated in Table II, our approach consistently outperforms all baselines, achieving faster and more reliable replanning across domains.

In addition, we directly applied the subgoal sequences and GNN model trained on the *Tower Construction* domain to the *Tool Use* domain to evaluate their robustness in the presence of new actions and objects. This is achieved by simply augmenting the original object set with (`tool`, `hook`). We observed that the subgoal sequences consistently guided the TAMP solver during replanning, while the GNN model reliably identified the temporally closest subgoals in the new domain. Notably, the object reduction strategy remained effective: it consistently minimized the number of objects considered for subgoals that did not require the use of the *hook*, while automatically reverting to full-environment planning for subgoals involving tool usage. As a result, we observed an increase in planning time in the *Tool Use* domain compared to *Tower Construction*, primarily due to the inclusion of all environmental objects when tool-based actions were necessary. These results demonstrate that the learned subgoal sequences and trained GNN model are transferable to structurally similar domains that introduce new objects or extend the action set. This generalization capability is crucial, as it allows for the continuous accumulation of domain-specific experience and incremental refinement of the subgoal sequences and GNN model in a lifelong learning setting—without altering the overall architecture of the proposed replanning framework.

D. Real-world Experiments

We conducted a real-world meal cooking experiment using a 7-DoF Franka robot arm equipped with a RealSense D415 camera fixed in the environment to capture the poses of environmental objects marked with visual markers, as shown in Figure 1. For this experiment, we utilized the subgoal sequences and the GNN model derived from *B3-Cooking* with objects in the YCB dataset. The task is simplified by

assuming ingredients are pre-washed and pre-cut, ready to be cooked directly if a feasible grasping and pouring for cooking path is found. However, challenges in non collision-free grasping due to blockers remain. We introduced an orange and a peach as additional ingredients that are non-essential for cooking the target meal in the initial environment configuration, corresponding to L2 disturbances. Additionally, we replaced the crackers box to obstruct the robot from cooking the meat in the can, as an L1 disturbance. The newly added orange and peach are never seen object type in the original tasks and used GNN model. We observed that our method found feasible plans for reacting to disturbances by predicting the correct subgoal and overall considering all the environmental objects as essential when disturbing the crackers box to block the manipulation of ingredients (taking around 2.87 seconds). However, they still managed to filter out non-essential objects when the crackers box does not block the grasping of essential ingredients and react to disturbances within around 0.83 second. Figure 1 shows representative screenshots of the task execution.

We also evaluated our framework in a PR2 cooking scenario in a simulated kitchen environment. Please refer to our supplementary video in our project webpage for more details and videos about our simulated and real-world experiments.

V. DISCUSSIONS

In this paper, we primarily focus on simple motion planning problems, such as picking and placing objects, in order to isolate and study the influence of planning horizon length and the number of environment objects on planning efficiency. For the tasks in Tables I and II, we define motion planning streams that can be solved in approximately 3 milliseconds. However, when the constraint graph search involves more than eight objects and a planning horizon exceeding 30, the computation time can surpass 30 seconds. Our results show that the proposed approach reduces such planning time by decreasing both the planning horizon and the number of objects involved. Even for tasks with complex motion planning subproblems, we expect our method to remain effective by limiting the number of motion planning subproblems encountered during replanning through planning horizon reduction.

While our method improves efficiency, our goal decomposition breaks the probabilistic completeness guarantee of the original PDDLStream solver when operating in replanning mode. In addition, our current computational distance metric approximates the computational complexity of planning problems under the assumption that motion planning

subproblems require similar planning time, which may not hold in complex scenes. Furthermore, during replanning, we currently consider only the computational time when selecting the closest subgoal, without accounting for actual execution time. In future work, it would be valuable to design improved computational distance metrics to enable more optimal subgoal selection. In addition, our current replanning strategy does not explicitly reason about failure cases and instead conducts replanning from scratch. A promising future direction is to include a failure reasoning module to generate more informative and efficient replanning strategies.

VI. CONCLUSION

In this work, we developed an efficient task and motion replanning approach that integrates our proposed problem decomposition learning method with classical TAMP solvers to enable efficient sequential multi-object manipulation in dynamic environments. We introduced techniques to learn subgoal sequences and computational distance function from demonstrations, which significantly reduce the planning horizon and the number of objects involved during replanning—resulting in substantial improvements in replanning efficiency. Through extensive numerical experiments, we also showed that task goals with more subgoals generally require longer planning times but are inherently more decomposable. This key observation motivated the incorporation of goal decomposition and computational distance learning to reduce the effective planning horizon, thereby enhancing the reactivity of TAMP solvers.

REFERENCES

- [1] Z. Yang, C. Garrett, T. Lozano-Perez, L. Kaelbling, and D. Fox, “Sequence-based plan feasibility prediction for efficient task and motion planning,” in *Proc. Robotics: Science and Systems (RSS)*, 2023.
- [2] F. Suárez-Ruiz, X. Zhou, and Q.-C. Pham, “Can robots assemble an ikea chair?” *Science Robotics*, vol. 3, no. 17, p. eaat6385, 2018.
- [3] V. N. Hartmann, A. Orthey, D. Driess, O. S. Oguz, and M. Toussaint, “Long-horizon multi-robot rearrangement planning for construction assembly,” *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 239–252, 2022.
- [4] Y. Du, O. Watkins, Z. Wang, C. Colas, T. Darrell, P. Abbeel, A. Gupta, and J. Andreas, “Guiding pretraining in reinforcement learning with large language models,” in *Proc. Intl Conf. on Machine Learning (ICML)*. PMLR, 2023, pp. 8657–8677.
- [5] N. Di Palo, A. Byravan, L. Hasenclever, M. Wulfmeier, N. Heess, and M. Riedmiller, “Towards a unified agent with foundation models,” in *Workshop on Reincarnating Reinforcement Learning at ICLR*, 2023.
- [6] 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.
- [7] M. Toussaint, “Logic-geometric programming: An optimization-based approach to combined task and motion planning,” in *Intl Joint Conf. on Artificial Intelligence IJCAI*, 2015, pp. 1930–1936.
- [8] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, “PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning,” in *Proc. of the Intl Conf. on Automated Planning and Scheduling*, vol. 30, 2020, pp. 440–448.
- [9] T. Silver, R. Chitnis, A. Curtis, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, “Planning with learned object importance in large problem instances using graph neural networks,” in *Proc. AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, 2021, pp. 11 962–11 971.
- [10] L. P. Kaelbling and T. Lozano-Perez, “Hierarchical task and motion planning in the now,” in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, May, p. 1470–1477.
- [11] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, and S. Savarese, “Neural task programming: Learning to generalize across hierarchical tasks,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. Brisbane, QLD: IEEE, May 2018, p. 3795–3802.
- [12] D. Xu, R. Martín-Martín, D.-A. Huang, Y. Zhu, S. Savarese, and L. F. Fei-Fei, “Regression planning networks,” *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, p. 1319–1329, 2019.
- [13] K. Elimelech, L. E. Kavraki, and M. Y. Vardi, “Extracting generalizable skills from a single plan execution using abstraction-critical state detection,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, May 2023, p. 5772–5778.
- [14] K. Elimelech, Z. Kingston, W. Thomason, M. Y. Vardi, and L. E. Kavraki, “Accelerating long-horizon planning with affordance-directed dynamic grounding of abstract strategies,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, May 2024, pp. 12 688–12 695.
- [15] S. Levit, J. Ortiz-Haro, and M. Toussaint, “Solving sequential manipulation puzzles by finding easier subproblems,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, May 2024, pp. 14 924–14 930.
- [16] B. Kim and L. Shimanuki, “Learning value functions with relational state representations for guiding task-and-motion planning,” in *Proc. Conference on Robot Learning (CoRL)*. PMLR, 2020, pp. 955–968.
- [17] D. Driess, O. Oguz, J.-S. Ha, and M. Toussaint, “Deep visual heuristics: Learning feasibility of mixed-integer programs for manipulation planning,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2020, pp. 9563–9569.
- [18] S. Park, H. C. Kim, J. Baek, and J. Park, “Scalable learned geometric feasibility for cooperative grasp and motion planning,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 545–11 552, 2022.
- [19] M. Khodeir, B. Agro, and F. Shkurti, “Learning to search in task and motion planning with streams,” *IEEE Robotics and Automation Letters*, vol. 8, no. 4, p. 1983–1990, Apr. 2023.
- [20] T. Migimatsu and J. Bohg, “Object-centric task and motion planning in dynamic environments,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 844–851, 2020.
- [21] J. Harris, D. Driess, and M. Toussaint, “FC³: Feasibility-based control chain coordination,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 13 769–13 776.
- [22] Y. Zhang, C. Pezzato, E. Trevisan, C. Salmi, C. H. Corbato, and J. Alonso-Mora, “Multi-modal MPPI and active inference for reactive task and motion planning,” *IEEE Robotics and Automation Letters*, vol. 9, no. 9, pp. 7461–7468, Sept. 2024.
- [23] T. Xue, A. Razmjoo, and S. Calinon, “D-LGP: Dynamic logic-geometric program for reactive task and motion planning,” in *Proc. IEEE Intl Conf. on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14 888–14 894.
- [24] N. Castaman, E. Pagello, E. Menegatti, and A. Pretto, “Receding horizon task and motion planning in changing environments,” *Robotics and Autonomous Systems*, vol. 145, p. 103863, 2021.
- [25] C. V. Braun, J. Ortiz-Haro, M. Toussaint, and O. S. Oguz, “RHH-LGP: Receding horizon and heuristics-based logic-geometric programming for task and motion planning,” in *Proc. IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2022, pp. 13 761–13 768.
- [26] Y. Zhang, T. Xue, A. Razmjoo, and S. Calinon, “Logic learning from demonstrations for multi-step manipulation tasks in dynamic environments,” *IEEE Robotics and Automation Letters*, vol. 9, no. 8, pp. 7214–7221, Aug. 2024.
- [27] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas, “A survey of sequential pattern mining,” *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54–77, 2017.
- [28] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, “Mining sequential patterns by pattern-growth: The prefixspan approach,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 11, pp. 1424–1440, 2004.
- [29] V. Myers, C. Zheng, A. Dragan, S. Levine, and B. Eysenbach, “Learning temporal distances: Contrastive successor features can provide a metric structure for decision-making,” in *Proc. Intl Conf. on Machine Learning (ICML)*, vol. 235. PMLR, Jul. 2024, pp. 37 076–37 096.
- [30] F. Di Giovanni, L. Giusti, F. Barbero, G. Luise, P. Lio, and M. M. Bronstein, “On over-squashing in message passing neural networks: The impact of width, depth, and topology,” in *Proc. Intl Conf. on Machine Learning (ICML)*. PMLR, 2023, pp. 7865–7885.