Rapid Replanning in Consecutive Pick-and-Place Tasks with Lazy Experience Graph

Tin Lai†,* and Fabio Ramos†,§

Abstract—In an environment where a manipulator needs to execute multiple consecutive tasks, the act of object manoeuvre will change the underlying configuration space, affecting all subsequent tasks. Previously free configurations might now be occupied by the manoeuvred objects, and previously occupied space might now open up new paths. We propose Lazy Tree-based Replanner (LTR*)—a novel hybrid planner that inherits the rapid planning nature of existing anytime incremental sampling-based planners. At the same time, it allows subsequent tasks to leverage prior experience via a lazy experience graph. Previous experience is summarised in a lazy graph structure, and LTR* is formulated to be robust and beneficial regardless of the extent of changes in the workspace. Our hybrid approach attains a faster speed in obtaining an initial solution than existing roadmap-based planners and often with a lower cost in trajectory length. Subsequent tasks can utilise the lazy experience graph to speed up finding a solution and take advantage of the optimised graph to minimise the cost objective. We provide proofs of probabilistic completeness and almost-surely asymptotic optimal guarantees. Experimentally, we show that in repeated pick-and-place tasks, LTR* attains a high gain in performance when planning for subsequent tasks.

I. INTRODUCTION

Common robotics applications involve the use of manipulator to manoeuvre objects from some initial locations to target locations [1]. Typically industrial robots in automated manufacturing systems [2] have tasks composed of reaching, grasping and placing motions, coupled with changing collision geometry during execution due to the attached and moved object [3]. Executing a task will likely modifies the underlying Configuration Space (C-space), which refers to the set of all possible robot configurations [4]. The modified C-space renders its previous motion plans invalid as the previously collision-free space might now be occupied by the moved objects. Therefore, a motion planner cannot assumes configurations remain valid between each consecutive task.

Sampling-based motion planners (SBPs) are a class of robust methods for motion planning [5]. In contrast, roadmap-based approaches are capable of querying motion plans for a different set of initial and target configurations [6] by utilising a persistent data structure. However, since objects are being manoeuvred during consecutive tasks, roadmap-based SBPs need to re-evaluate all of their edge validity because the underlying C-space changes in-between each task. The re-evaluation lowers the effectiveness of its multi-query property since it requires more time to build a sufficient roadmap when compared to other methods [7]. On the other hand, existing tree-based SBPs have the properties of creating motion plans that inherently have lower cost and take less amount of time [6]. However, this class of methods needs to create a new motion plan from scratch for every new task. Since the manipulator always operates under the same environment (with objects moved by the manipulator itself), there are often many similar structure in-between each planning instance, for example, in consecutive pick-and-place tasks.

The pick-and-place problem differentiates itself from other planning problems by continuously operating in the same environment with spatial changes that are localised. Localised changes refer to objects moved due to manipulation by the robot itself. These changes tend to be concentrated within certain regions and in the vicinity of the manipulator’s gripper. However, there is often valuable information collected from past planning problems that remain helpful in subsequent tasks. Our proposed model remediates this issue by providing a framework that allows the planner to plans for optimal trajectories while utilising past connections previously known to be valid to bootstrap future solutions.

We propose a hybrid planner that tackles the consecutive tasks problem. Unlike existing multi-query SBPs, our planner does not require any pre-processing graph construction step. Instead, a rapid tree-based method is used to begin the planning process while simultaneously building a lazy experience graph to capture the connectivity. There is minimal computational overhead involved since no collisions are performed yet. In subsequent tasks, the lazy graph will then persist as a sparse structure that captures the previous experience under the same environment. When the trees are close to the existing experience graph, the graph will be utilised to bootstrap the solution with the shortest path search. Our planner has comparable performance against other tree-based methods in completely new environments. However, in subsequent tasks, our planner utilises the lazy experience graph in a hybrid manner combining tree-based and graph-based approaches,
often resulting in faster initial solutions and lower length cost.

II. RELATED WORKS

Sampling-based motion planners take a probabilistic approach to the planning problem. There exist works that attempt to address the problem of being efficient in planning within the same environment. For example, PRM [9] performs planning within the same environment by saving previous requests in a graph-like structure. The star-varients of SBPs often denote their asymptotic optimality nature in regards to minimising some cost functions. Learning techniques like modelling the success probability [10] or with a MCMC sampling scheme [11] are possible approaches to improve performance. Multiple tree or graph-like structures can also be utilised for efficient exploration. For example, bidirectional trees [12] or multi-tree approach [8] for object picking [13] to speed up the planning process by exploring different regions using a pre-computed database. Combining tree-based and roadmap-based approaches had been explored in [14] which creates multiple trees in parallel but sacrifices solution speed due to its space-filling approach.

Since obstacles within the world-space would be altered during the course of the consecutive tasks problem, experience-based planners like PRM cannot directly reuse its roadmap for subsequent tasks. Lazy-PRM* [15] was designed to reduce the number of collision checking within its graph, but it can be altered to suit the need of the pick-and-place domain. It lazily evaluates the validity of edges within the graph which saves computational time. Elastic roadmaps [16] had been used in mobile manipulators, which reuse manipulator planning in different 2D spatial locations. However, the approach does not benefit in fixed-frame manipulators. E-graph [17] focuses on utilising heuristics to stores previous solution trajectories into a structure to accounts for environment changes. LPA* [18] maintain a structure after the first search, and reuses previous search tree that are identical to the new one. The underlying changes in workspace can also be modelled as subset constrain manifold to aid solving subsequent tasks [19].

Consecutive tasks problem typically arises in Task and Motion Planning which can encapsulate the execution of tasks as skills [20] via learning state transitions [21]. Techniques that focus on improve sampling efficiency can be modified to restrict the sampling region [22] or use a learned sampling distribution [23]–[27] to formulate a machine learning approach to learn from previous experience. An experience-based database can help to improve planning efficiency. For example, the Lightning [28] and Thunder [29] framework uses some database structure to store past trajectories, but does not generalise solutions in-between trajectories.

III. LAZY TREE-BASED REPLANNER

In this section, we begin by formulating the consecutive tasks with objects manipulation problem under optimal motion planning. Then, we conceptualise the Lazy Tree-based Replanner framework and discuss the use of the Lazy Experience Graph. Finally, we will analyse the algorithm tractability and completeness of the proposed algorithm.

A. Formulation

Definition 1 (Object Manipulation Task): An object manipulation task $\tau_i$ is a tuple $(\Theta_{\tau_i}, x_{\text{target},i})$ composed of an object $\Theta_{\tau_i}$ in the workspace $\omega$, and a target coordinate $x_{\text{target},i} \in \mathbb{SE}(3)$. Let $x(\Theta_{\tau_i}) \in \mathbb{SE}(3)$ denotes the current coordinate and orientation of the object $\Theta_{\tau_i}$. Then, an object manipulation task $\tau_i$ refers to moving $\Theta_{\tau_i}$ from its initial pose $x(\Theta_{\tau_i})$ to its target pose $x_{\text{target},i}$ within the time budget.

The repeated tasks scenario refers to a manipulator completing several instances of tasks (definition 1) while avoiding collision with the environment. Note that while the environment is static, the manipulator will modify the underlying $C$-space throughout the consecutive tasks due to the moved objects in $\omega$. In the following, $\sigma : [0, 1] \to C_{\text{free}}$ is a parameterised trajectory where $\sigma(0)$ and $\sigma(1)$ refers to the start and end of the trajectory in the configuration space.

Problem 1 (Consecutive Manipulation Tasks Problem): Given the set of configuration space $C \subseteq \mathbb{R}^d$, the set of free space $C_{\text{free}} \subseteq C$, a set of objects manipulation tasks $T = \{\tau_1, \ldots, \tau_n\}$, and a function $f_q : \omega \to C_{\text{free}}$ that maps any given object $\Theta_{\tau_i}$ from its current pose to a reachable and feasible grasp configuration. Formally, find a trajectory $\sigma$ such that all $\tau_i \in T$ are completed according to definition 1 using $f_q$ for grasping each $\Theta_{\tau_i}$, where $\forall \theta \in [0, 1], \sigma(\theta) \in C_{\text{free}}$.

Problem 2 (Optimal Planning): Let $\Gamma(C_{\text{free}})$ denotes the set of all possible trajectories in $C_{\text{free}}$. Given $C, C_{\text{free}}, T$ with specified task execution order, and a cost function $\mathcal{L} : \sigma \to [0, \infty)$; find a solution trajectory $\sigma^*$ that incurs the lowest cost. Formally, find trajectory $\sigma^*$ such that

$$\mathcal{L}(\sigma^*) = \min_{\sigma \in \Gamma(C_{\text{free}})} \mathcal{L}_C(\sigma)$$

and it satisfies problem 1

B. Lazy Tree-based Replanner

We propose Lazy Tree-based Replanner (LTR*)—an incremental, asymptotic optimal sampling-based planner capable of rapidly replans for executing consecutive tasks. LTR* unifies tree-based and roadmap-based methods by utilising the rapid sampling procedure with bidirectional trees whilst using a lazy experience graph to captures the connectivity in $C$-space for future tasks. Our target domain—consecutive object manipulation planning—is inherently a multi-query problem. However, unlike typical multi-query planners, LTR* avoids pre-processing overhead by incrementally constructs its lazy-experience graph using a single-query motion planner. When LTR* is first planning under a new environment, the procedure is similar to that of a single-query SBP. LTR* first utilises bidirectional trees [30] to creates an initial motion plan, which is a state-of-the-art asymptotic optimal planner that often outperforms other tree-based planners. During the planning procedure, LTR* captures the connectivity of $C$-space with a lazy experience graph similar to that of a Lazy-PRM*.

The graph is for capturing the connectivity in the
current environment, and the validity of the edges will not be evaluated yet. In subsequent tasks, LTR* outperforms other methods by bootstrapping solution with its experience graph. Rapid tree-based methods are used to expand outward from its initial and target configurations. Then, when the tree structure is close to its previous lazy experience graph, it will perform a lazy shortest path search within the graph to connect both trees. Since the lazy graph captures C-space connectivity from its previous tasks, most connections will be valid even if there is a modification in the C-space due to the moved objects. As a result, LTR* can rapidly replan for subsequent tasks, e.g. in consecutive pick-and-place tasks, by utilising its experience graph.

C. Implementation

Algorithm 1 illustrates LTR*’s overall algorithmic aspect. Line 17 begins the consecutive pick-and-place tasks problem by first initialising its persistent lazy experience graph \( \mathcal{G} \). Incoming tasks are processed in lines 18 to 23 where the execution order of tasks is not of the scope of this problem. Each planning request is sent to the PlanTrajectory subroutine, where \( f_g : \omega \rightarrow C_{\text{free}} \) is a function that maps from object to a possible grasp configuration. Notice that the underlying C-space structure changes (i.e. changing collision geometry) after each pickup and place operation due to the object being attached to the manipulator and moved objects in \( \omega \). PlanTrajectory subroutine takes two configurations as its planning input, along with the persistent graph that saves its prior experience. When \( \mathcal{G} \) is empty, LTR* behaves similar to that of a bidirectional RRT* as lines [10] to [14] would not be entered. While LTR* is growing its trees, it also saves the created vertices and edges information in a temporary graph \( \mathcal{G}' \) (line 7) which would be merged with \( \mathcal{G} \) at the end of a task. There are no additional computation overheads when building \( \mathcal{G}' \) except that of the nearest neighbour search (NNS) in a k-d tree structure since collision checking is not performed yet (Left of Figure 2). The search can be further optimised out by utilising information from GrowTree (line 6) since a typical tree-based planner also uses NNS, and both planners operate on the same set of vertices. Similar to existing bidirectional tree approaches (i.e., Alg. 1 in [30]) GrowTree subroutine samples a random configuration and expands the current tree \( T \) while attempting to connect towards the other tree. When the NNS found some \( v_{\text{new}} \) within \( \epsilon_n \) of \( \mathcal{G} \) during the tree expansion, LTR* will attempts to connect \( T \) with \( \mathcal{G} \) (lines [10] to [12]). When both trees are connected to \( \mathcal{G} \), LTR* can bootstrap a solution by performing a shortest path search in \( \mathcal{G} \) at the two connected vertices. Edges will be lazily validated in the typical Lazy PRM* fashion and saved for subsequent search [15]. If the shortest path returns a valid solution at the two connected vertices, then LTR* has successfully found a solution that connects from the initial configuration, through \( T_{\text{initial}}, \mathcal{G} \) and \( T_{\text{target}} \) to the target configuration (lines [13] and [14]). In Figure 2 the right illustration provides an overview of the above scenario.

LTR* follows the approach in [6] which uses the connection radius \( \epsilon_n \) as a function of the number of node \( n \in \mathbb{N} \), where \( \epsilon_n = \epsilon(n) := \gamma (\log n/n)^{1/d}, \gamma > \gamma^* = 2(1 + 1/d)^{1/d}(\mu(C_{\text{free}})/\xi_d)^{1/d}, \xi_d \) is the volume of the unit ball in \( d \)-dimensional Euclidean space, and \( \mu(X) \) denotes the Lebesgue measure of a set \( X \). In practice, the lazy experience graph built by LTR* tends to inherit the tree-based planners’ property of concentrating more edges closer to the initial and target configurations (Figure [1]). While such a graph is not as diverse as that of a Lazy-PRM*, it is, in fact, more beneficial under a wide range of manipulator and environment setups since most pickup and place tasks will be concentrated at some central location. As a result, the lazy experience graph helps to bootstrap solutions that often exhibit lower costs than Lazy-PRM*.

D. Algorithmic tractability

LTR* uses experience graph from past plannings to explores C-space, which is guaranteed to help the current planning if its prior knowledge is viable in the current settings. Furthermore, LTR* is probabilistic complete and will asymptotically converge its solution to the optimal one. In the following, we will use \( V_g(R) \) to denote the \( d \)-dimensional volume of a hypersphere of radius \( R \), and \( V(q) \) to denote the visibility set of \( q \) which represents the region of \( C_{\text{free}} \) visible from some \( q \in C_{\text{free}} \).
When \( \text{LTR}^* \) grows its trees (top layer) from its initial configuration (green node), it simultaneously builds a lazy experience graph (bottom layer). Solid orange edges in the bottom layer retain the checked validity from its original tree structure, whereas the blue dotted edges are added without checking validity yet (hence there are some invalid connections in the example given). (Right) Illustration of \( \text{LTR}^* \) using its previously built lazy experience graph (red graph) to speed up the solution. The two cyan trees on either side are built for the current pick-and-place task, and when those trees are in close proximity to the experience graph, it can return a fast solution by performing a shortest path graph-search at the connected nodes (resulting in yellow solution trajectory). Note that the simultaneously built experience graphs for this task are not shown (i.e. the orange graph at the bottom left layer) for clarity. The red (past experience) and orange (newly built experience) graphs will be merged for subsequent tasks.

**Definition 2 (Connected free space):** \( C_{\text{free}}' \subseteq C_{\text{free}} \) is said to be a connected subset of free space if \( \forall n \in \mathbb{N}, m > 2, \exists \varepsilon > \varepsilon_n \) for \( n \geq m \) such that

\[
\forall q \in C_{\text{free}}', \quad \mu(\mathcal{V}(q)) \geq V_d(\varepsilon).
\]

(2)

Eq. (2) states that each configuration in \( C_{\text{free}}' \) connects to both \( \mathcal{V} \) and \( \mu \) denote the visibility set of \( q \) and \( \mu \) denotes the visibility set of \( q \) containing at least \( \varepsilon \) radius of surrounding free space which will eventually surpass the shrinking radius \( \varepsilon_n \) when \( n \geq m \), where \( m \) is environment specific.

**Theorem 1 (Joining of Lazy Experience Graph):** Let free space \( C_{\text{free}} \subseteq C \) be a bounded set. Consider \( C_{\text{free}}' \subseteq C_{\text{free}} \) to be a connected subset of the free space \( C_{\text{free}} \) according to definition 2. Given that \( \mathcal{G} \neq \emptyset \). If \( \mathcal{T}_{\text{initial}}, \mathcal{T}_{\text{target}} \subseteq C_{\text{free}}' \), \( \mathcal{T}_{\text{initial}} \) connects \( \mathcal{G} \) and \( \mathcal{T}_{\text{target}} \) connects \( \mathcal{G} \) with probability one.

**Proof:** From Alg. 1 lines 10 to 14, we add each new node from \( \mathcal{T}_{\text{initial}} \) that are reachable and within \( \varepsilon_n \) radius. The event of joining the tree \( \mathcal{T}_{\text{initial}} \), and \( \mathcal{G} \) occurs if a newly sampled configuration \( q_n \) exists for each pair \( q \) and \( q \in \mathcal{G} \) where \( q_n \) lies within a region that is visible and connectable to both \( \mathcal{T}_{\text{initial}} \) and \( \mathcal{G} \). The connectable region \( S \) is given by

\[
S = \bigcup_{q \in \mathcal{T}_{\text{initial}}} \mathcal{V}_n(q) \cap \bigcup_{q \in \mathcal{G}} \mathcal{V}_n(q)
\]

where

\[
\mathcal{V}_n(q) := \{ q_i \in \mathcal{V}(q) \mid \|q_i - q\| \leq R \}
\]

denotes the visibility set of \( q \) bounded by the hypersphere of radius \( R \). The entire experience graph and the tree will occupy \( \mu^\mathcal{G} \) and \( \mu^\mathcal{T}_{\text{initial}} \) amount of volume in \( C_{\text{free}}' \) where

\[
\mu^\mathcal{G} = \bigcup_{q \in \mathcal{G}} \mu(\mathcal{V}_n(q)) \quad \text{and} \quad \mu^\mathcal{T}_{\text{initial}} = \bigcup_{q \in \mathcal{T}_{\text{initial}}} \mu(\mathcal{V}_n(q)).
\]

Eq. (2) states that each \( q \) has at least \( V_d(\varepsilon) > V_d(\varepsilon_n) \) free space to be connected with surrounding configurations with \( \varepsilon_n \) connection radius. If the lazy experience graph \( \mathcal{G} \) is non-empty (from past planning instances), (2) further ensures that the connectable space from \( \mu^\mathcal{G} \) and \( \mu^\mathcal{T}_{\text{initial}} \) has non-zero volume, and hence non-zero probability of begin sampled by a uniform sampler. Since the tree \( \mathcal{T}_{\text{initial}} \) is growing in \( C_{\text{free}}' \), the tree will eventually reach all visible configurations in \( C_{\text{free}}' \) as the number of nodes approach infinity (see [30]). Therefore, the probably that \( \mathcal{T}_{\text{initial}} \) joins with \( \mathcal{G} \) is given by

\[
\lim_{n \to \infty} P(q_n \in S) = 1,
\]

where \( q_n \) is the \( n \)th sample of configuration for \( \mathcal{T}_{\text{initial}} \). The same proof can be applied to the joining of \( \mathcal{T}_{\text{target}} \) and \( \mathcal{G} \). \( \text{LTR}^* \) attains probabilistic completeness as its number of uniform random samples approaches infinity.

**Theorem 2 (Probabilistic Completeness):** \( \text{LTR}^* \) inherits the same probabilistic completeness of bidirectional \( \text{RRT}^* \) in each pick-and-place task. Let \( q_{\text{initial}}^{\tau_i} \) and \( q_{\text{target}}^{\tau_i} \) denotes the initial and target configuration provided in each task \( \tau_i \in \mathcal{T} \), and \( V_{\tau_i}^{\mathcal{G}} \) denotes the set of configurations reachable from \( q_{\text{initial}}^{\tau_i} \) through connected edges for \( \tau_i \). Given that a solution exists for each pair \( q_{\text{initial}}^{\tau_i} \) and \( q_{\text{target}}^{\tau_i} \), \( \text{LTR}^* \) is guaranteed to eventually find a solution. Formally,

\[
\forall \tau_i \in \mathcal{T}, \quad \lim_{t \to \infty} \mathbb{P}(\{V_{\tau_i}^{\mathcal{G}} \cap \{q_{\text{target}}^{\tau_i}\} \neq \emptyset\}) = 1.
\]

(7)

**Proof:** If a solution exists for task \( \tau_i \), then \( q_{\text{initial}}^{\tau_i} \) and \( q_{\text{target}}^{\tau_i} \) must lies in the same connected free space \( C_{\text{free}}' \). If part of \( \mathcal{G} \) also lies in the the same \( C_{\text{free}}' \), \( q_{\text{initial}}^{\tau_i} \) and \( q_{\text{target}}^{\tau_i} \) are guaranteed to connects to \( \mathcal{G} \) (theorem 1). In the event that the existing lazy experience graph does not lies in \( C_{\text{free}}' \), \( \text{LTR}^* \) behaves the same as that of a bidirectional \( \text{RRT}^* \) because no connections would be made to \( \mathcal{G} \) and \( \text{LTR}^* \) would be reduced to its internal planner’s subroutine (Alg. 1 line 10 to line 14 would not be entered). Therefore, \( \text{LTR}^* \) attains
Table I

Numerical results for the Panda, Jaco and TX90 arm scenarios (μ ± σ).

|                   | # Task | Panda arm | Jaco arm | TX90 arm |
|-------------------|--------|-----------|----------|----------|
|                   |        | Time (s)  | Cost     | Time (s) | Cost     |
|                   |        |           |           |           |           |
| LTR*              | Pick   | 1st       | 2.36 ± 1.42 | 9.13 ± 3.76 | 8.72 ± 3.85 |
|                   |        | 8th       | 0.38 ± 0.25 | 10.17 ± 4.16 | 5.12 ± 3.73 |
|                   | Place  | 1st       | 1.23 ± 1.14 | 8.45 ± 2.97 | 12.76 ± 7.21 |
|                   |        | 8th       | 0.18 ± 0.16 | 9.91 ± 0.58 | 4.9 ± 4.91 |
| Lazy-PRM*         | Pick   | 1st       | 0.91 ± 1.26 | 22.91 ± 15.82 | 10.67 ± 5.75 |
|                   |        | 8th       | 1.31 ± 0.36 | 23.86 ± 23.83 | 3.76 ± 7.81 |
|                   | Place  | 1st       | 1.83 ± 2.58 | 24.73 ± 22.18 | 9.93 ± 11.38 |
|                   |        | 8th       | 0.85 ± 0.41 | 20.83 ± 17.75 | 7.84 ± 9.13 |
| rRT-Connect*      | Pick   | 1st       | 1.47 ± 2.49 | 9.86 ± 4.78 | 8.18 ± 9.74 |
|                   |        | 8th       | 2.11 ± 2.25 | 10.18 ± 4.17 | 8.74 ± 9.13 |
|                   | Place  | 1st       | 1.32 ± 0.98 | 8.91 ± 4.16 | 7.99 ± 9.79 |
|                   |        | 8th       | 1.28 ± 2.1 | 9.17 ± 5.83 | 9.36 ± 8.67 |
| E-Graph           | Pick   | 1st       | 2.76 ± 1.01 | 17.80 ± 10.27 | 13.86 ± 5.86 |
|                   |        | 8th       | 1.77 ± 2.92 | 15.91 ± 16.09 | 15.72 ± 8.19 |
|                   | Place  | 1st       | 2.69 ± 1.38 | 15.26 ± 11.58 | 10.99 ± 11.27 |
|                   |        | 8th       | 1.82 ± 3.27 | 12.39 ± 8.11 | 9.36 ± 8.67 |

Fig. 3. (Top) Jaco and (bottom) TX90 arm scenarios.

Theorem 3: Let $n_i^{LTR*}$ and $n_i^{bRRRT*}$ be the number of uniformly random configurations sampled at iteration $i$ for LTR* and bidirectional RRT* respectively. There exists a constant $\phi \in \mathbb{R}$ such that

$$\lim_{i \to \infty} \mathbb{E}[n_i^{LTR*} / n_i^{bRRRT*}] \leq \phi. \quad (8)$$

Proof: As stated in theorem 1, $T_{\text{initial}}$ and $T_{\text{target}}$ has non-zero probability of adding new nodes at each iteration, hence as $i \to \infty$ both trees will have infinite samples to refine its branches. In the LTR* algorithm, it uses the experience graph $G$ to bootstrap solution. However, the internal structure of the trees are maintained separately (Alg. 1 line 6), and edges within the trees are not added to the trees when using $G$ to bridge the gap between $T_{\text{initial}}$ and $T_{\text{target}}$. Therefore, the asymptotic behaviour of the growth of $T_{\text{initial}}$ and $T_{\text{target}}$ in LTR* approaches bidirectional RRT* as $i \to \infty$.

Although LTR* uses the (possibly non asymptotic optimal compliant) $G$ to bootstraps the connection between $T_{\text{initial}}$ and $T_{\text{target}}$, LTR* guarantees the solution it returns will asymptotically converges to the optimal solution.

Theorem 4 (Asymptotic optimality): Let $\sigma_i^*$ be LTR*'s solution at iteration $i$, and $c^*$ be the minimal cost for problem $P$. If a solution exists, then the cost of $\sigma_i$ will converge to the optimal cost almost-surely. That is,

$$\mathbb{P}\left(\lim_{i \to \infty} c(\sigma_i) = c^*\right) = 1. \quad (9)$$

Proof: Since LTR* uses experience graph that are not optimised against the current task’s $T_{\text{initial}}$ and $T_{\text{initial}}$, the solution trajectory $\sigma_i$ that LTR* returns at iteration $i$ does not satisfy the asymptotic criteria specified in [6]. However, even after $T_{\text{initial}}$ and $T_{\text{target}}$ are joined with $G$ (i.e. after obtaining fast initial solution), LTR* only adds $v \in G$ within $\epsilon_n$ to $\mathcal{T}$ (Alg. 1 line 10) sequentially and with adequate rewrite procedure (Alg. 1 line 12) within the internal tree structure. Therefore, as the asymptotic behaviour of LTR*'s approaches that of a bidirectional RRT* (theorem 3) after $T_{\text{initial}}$ and $T_{\text{target}}$ connects without the bridging of $G$, the cost of $\sigma_i$ that LTR* returns converges to the optimal cost almost surely.

IV. Experiments

We experimentally evaluate the versatility and performance of LTR* by testing for speed and cost in multiple environments. Since different types of manipulators exhibit distinct C-space, which will impact the underlying structures, we conduct our experiment on three different types of manipulators to assess LTR*'s robustness. The simulated environments are depicted in Figs. [1] and [3]. The manipulators used are the PANDA, Kinova Jaco and TX90 arm with an attached PR2 gripper. We first describe the environments and experimental setup, then provide a comparison against other state-of-the-art sampling-based planners under the same environment.

The scenario’s objective is to manoeuvre all of the objects $\Theta_i$ from their randomised starting location to their target location $\Theta_{target,i}$. The starting and target regions are randomly generated on some predefined surface for each scenario. There exist eight objects in each scenario, often on the surface of tables or cupboards. The objective of each task is to transfer an object from a starting location to a target location while avoiding collisions. We implement LTR* under the OMPL framework, and integrates it with the MoveIt! and Klampt simulator in Python [31]. The experiment is composed of 8 tasks, each of which requires a planning procedure for picking up and placing an object at some pre-generated random location. The manipulator will need to go back and forth through a constantly changing region, resulting in 16 motion trajectory

Probalistic completeness and its asymptotic behaviour is same as that of a bidirectional RRT*.

LTR* uses an experience graph to bootstrap fast solution trajectory; hence, unlike RRT* variants, LTR* does not possess the anytime property of all tree branches always exhibit the current optimal route. However, LTR* inherits asymptotic optimality where the solution trajectory will converges to the optimal trajectory almost surely.

**Theorem 3:** Let $n_i^{LTR*}$ and $n_i^{bRRRT*}$ be the number of uniformly random configurations sampled at iteration $i$ for LTR* and bidirectional RRT* respectively. There exists a constant $\phi \in \mathbb{R}$ such that

$$\lim_{i \to \infty} \mathbb{E}[n_i^{LTR*} / n_i^{bRRRT*}] \leq \phi. \quad (8)$$

**Proof:** As stated in theorem 1, $T_{\text{initial}}$ and $T_{\text{target}}$ has non-zero probability of adding new nodes at each iteration, hence as $i \to \infty$ both trees will have infinite samples to refine its branches. In the LTR* algorithm, it uses the experience graph $G$ to bootstraps solution. However, the internal structure of the trees are maintained separately (Alg. 1 line 6), and edges within the trees are not added to the trees when using $G$ to bridge the gap between $T_{\text{initial}}$ and $T_{\text{target}}$. Therefore, the asymptotic behaviour of the growth of $T_{\text{initial}}$ and $T_{\text{target}}$ in LTR* approaches bidirectional RRT* as $i \to \infty$.

Although LTR* uses the (possibly non asymptotic optimal compliant) $G$ to bootstraps the connection between $T_{\text{initial}}$ and $T_{\text{target}}$, LTR* guarantees the solution it returns will asymptotically converges to the optimal solution.

**Theorem 4 (Asymptotic optimality):** Let $\sigma_i^*$ be LTR*'s solution at iteration $i$, and $c^*$ be the minimal cost for problem $P$. If a solution exists, then the cost of $\sigma_i$ will converge to the optimal cost almost-surely. That is,

$$\mathbb{P}\left(\lim_{i \to \infty} c(\sigma_i) = c^*\right) = 1. \quad (9)$$

**Proof:** Since LTR* uses experience graph that are not optimised against the current task’s $T_{\text{initial}}$ and $T_{\text{initial}}$, the solution trajectory $\sigma_i$ that LTR* returns at iteration $i$ does not satisfy the asymptotic criteria specified in [6]. However, even after $T_{\text{initial}}$ and $T_{\text{target}}$ are joined with $G$ (i.e. after obtaining fast initial solution), LTR* only adds $v \in G$ within $\epsilon_n$ to $\mathcal{T}$ (Alg. 1 line 10) sequentially and with adequate rewrite procedure (Alg. 1 line 12) within the internal tree structure. Therefore, as the asymptotic behaviour of LTR*'s approaches that of a bidirectional RRT* (theorem 3) after $T_{\text{initial}}$ and $T_{\text{target}}$ connects without the bridging of $G$, the cost of $\sigma_i$ that LTR* returns converges to the optimal cost almost surely.
execution. LTR*, RRT-Connect*, Lazy-PRM* and E-Graph are tested in this experiment. The experiment for each planner is repeated 10 times each, as a total of 160 trajectories for each scenario. The lazy data structure of each planner is kept between each task, except bidirectional RRT* since it does not allow invalidating edges. The data structure of each planner is initially empty, except for E-Graph, where we first bootstrap its data structure with 5 uniformly distributed goals. Experimental results obtained is shown in table 1. Results from the 1st and 8th tasks are included to highlight the performance gain by exploiting the consecutive nature of the problem. LTR* and Lazy-PRM* operate at a similar speed on their 1st task and consistently improve their time metric (time required to obtain first solution) by the 8th task via reusing their stored graph; whereas RRT-Connect* needs to recompute its entire tree from scratch and does not improve its speed. E-Graph requires an initial bootstrap as it uses 3D Dijkstra and inverse kinematics to facilitate planning, which in average requires more than 30 seconds to obtain an initial solution in the PANDA environment with an empty graph. E-Graph gradually improves its time-to-solution as it stores more solution, but with a degraded solution cost. Moreover, E-Graph cannot provide any completeness and optimal convergence guarantee as it operates in workspace. Solutions returned by RRT-Connect* always consists of lower cost due to its tree-based nature of expanding configurations directly from the initial and target configurations, which is also true for LTR* since LTR* also bootstraps its lazy experience graph with trees. As a result, LTR* behaves like a hybrid in-between the other two, inheriting the benefit of graph reusability and low-cost solution nature. In Figure 4 we visually display results from the PANDA arm as boxplots of time and cost across the evolutions of tasks 1 to 8. The x-axis is divided into eight sections; each corresponds to picking up an object and placing it at the dedicated location. LTR* behaves similar to RRT-Connect* on the first task when the lazy experience graph has not been built yet (top of Figure 4). This is because the subroutine as depicted in Alg. 1 will not be entered on the first task (empty graph G). However, in subsequent tasks, LTR* can return a much faster solution by utilising the lazy experience graph to bootstrap an initial solution. The result is similar to that of a Lazy-PRM* where it can reuse its persistent data structure in subsequent tasks. Figure 5 depicts the same phenomenon with x-axis indicating LTR*’s benefit when the number of task increases. Unlike Lazy-PRM*, LTR* uses a tree-based method to explore and connects to its persistent graph. Tree-based methods tend to return a trajectory with a much lower cost due to the locality when growing the trees outwards from their initial and target configurations. For example, Lazy-PRM* had spent a substantial amount of time in the PANDA task creating nodes that are not critical to the optimal trajectory (Figure 4). In contrast, tree-based methods can directly bias its tree towards the start or target configuration. Therefore, LTR* can achieve a much lower cost overall than Lazy-PRM* (Bottom of Figure 4).

Overall the proposed method LTR* consistently outperforms other baselines on consecutive pick-and-place tasks, where it has the benefit of rapid planning speed of the bidirectional RRT* and the reusability of Lazy-PRM*. Interestingly, while there exist certain overheads for LTR* when it checks for connections between the trees $T_{initial}$, $T_{target}$ and the graph $G$, the quantitative results illustrate that the overall speeds-up overweight the overhead. The hybrid LTR* grows a tree

Fig. 4. Experimental results obtained from the PANDA consecutive pick-and-place tasks, to illustrate the evolution of LTR*’s performance after performing multiple tasks. The blue and red boxes correspond to the pickup and place task, respectively. Each planner has to plan a trajectory to pickup the object and place at the target location to complete each task. Boxplots of (Top) time to obtain initial solution and (Bottom) the final cost (length) of the trajectory.

Fig. 5. Experimental results for the multiple object picking tasks in PANDA problem setup (line and shaded region depict median ± 25% quartiles).
rapidly to the existing experience graph for a fast solution, and it will keep refining the tree when there is still time budget remained to refine its solution. In a life-long setting where the robot arm needs to execute an indefinitely long sequence of tasks, the lazy experience graph might become arbitrarily dense. An overly dense graph helps to ensure completeness and optimality guarantees; however, performing graph search in such a graph might degrade performance. One possible strategy to mitigate the issue is via witness set which helps to maintain a sparse data structure by pruning nodes from the graph. This strategy only ensures probabilistic ϵ-robust completeness; however, it can help control the lazy experience graph’s growth over the life-long setting.

V. Conclusion

We proposed a hybrid planner as a method for planning specifically in consecutive object manipulation tasks. The challenge lies in the modified configuration space in-between each task due to the moved objects and the need to rapidly replans while achieving low-cost trajectory. Our proposed method utilises a lazy experience graph to keep track of past data while performing tree-based planning to rapidly expands the search for low-cost trajectory. As a result, we are able to retain valuable information in-between each task, while achieving low trajectory cost that is commonly available in a tree-based planner. Such a hybrid method capitalise the rapid expansion property from tree-based SBPs, and the reusability from the roadmap-based SBPs. In the future, we can better exploit the change-of-obstacle locality by learning a sampling distribution that concentrates in “active regions”, which can then focus more on regions that require replanning.

References

[1] T. Lozano-Pérez, J. L. Jones, E. Mazer and P. A. O’Donnell, ‘Task-level planning of pick-and-place robot motions,’ Computer, vol. 22, no. 3, pp. 21–29, 1989.
[2] S. S. Perumal and N. Jawahar, ‘Automated Trajectory Planner of Industrial Robot for Pick-and-Place Task,’ International Journal of Advanced Robotic Systems, vol. 10, no. 2, p. 100, 2013.
[3] Y. Yang, W. Merkt, V. Ivan and S. Vijayakumar, ‘Planning in time-configuration space for efficient pick-and-place in non-static environments with temporal constraints,’ in 2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids), IEEE, 2018, pp. 1–9.
[4] M. Elbanhawi and M. Simic, ‘Sampling-based robot motion planning: A review,’ IEEE Access, vol. 2, pp. 56–77, 2014.
[5] L. E. Kavraki, M. N. Kolountzakis and J.-C. Latombe, ‘Analysis of probabilistic roadmaps for path planning,’ in Proceedings of IEEE International Conference on Robotics and Automation, vol. 4, IEEE, 1996, pp. 3020–3025.
[6] S. Karaman and E. Frazzoli, ‘Sampling-based algorithms for optimal motion planning,’ The International Journal of Robotics Research, vol. 30, no. 7, pp. 846–894, 2011.
[7] F. Burget, M. Bennewitz and W. Burgard, ‘BI 2 RRT*: An efficient sampling-based path planning framework for task-constrained mobile manipulation,’ in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2016, pp. 3714–3721.
[8] T. Lai, F. Ramos and G. Francis, ‘Balancing Global Exploration and Local-connectivity Exploitation with Rapidly-exploring Random disjointed-Trees,’ in Proceedings of The International Conference on Robotics and Automation, IEEE, 2019, pp. 5537–5543.
[9] L. E. Kavraki, P. Svestka, J. C. Latombe and M. H. Overmars, ‘Probabilistic roadmaps for path planning in high-dimensional configuration spaces,’ IEEE Transactions on Robotics and Automation, IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 566–580, 1996.
[10] T. Lai and F. Ramos, ‘Adaptively Exploits Local Structure with Generalised Multi-Trees Motion Planning,’ IEEE Robotics and Automation Letters, vol. 7, no. 2, pp. 1111–1117, 2022.
[11] T. Lai, P. Morere, F. Ramos and G. Francis, ‘Bayesian Local Sampling-Based Planning,’ IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 1954–1961, 2020.
[12] J. J. Kuffner and S. M. LaValle, ‘RRT-connect: An efficient approach to single-query path planning,’ in Proceedings of IEEE International Conference on Robotics and Automation, vol. 2, IEEE, 2000, pp. 995–1000.
[13] E. Littlefield, S. Zhu, H. Kourtvet, Z. Psarakis, R. Shome, A. Kimmel, A. Dobson, A. F. De Souza and K. E. Bekris, ‘Evaluating end-effector modalities for warehouse picking: A vacuum gripper vs a 3-finger underactuated hand,’ in 2016 IEEE International Conference on Automation Science and Engineering (CASE), IEEE, 2016, pp. 1190–1195.
[14] E. Plaku, K. E. Bekris, B. Y. Chen, A. M. Ladd and L. E. Kavraki, ‘Sampling-based roadmap of trees for parallel motion planning,’ IEEE Transactions on Robotics, vol. 21, no. 4, pp. 597–608, 2005.
[15] K. Hauser, ‘Lazy collision checking in asymptotically-optimal Motion planning,’ in 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2015, pp. 2951–2957.
[16] Y. Yang and O. Brock, ‘Elastic roadmaps—motion generation for autonomous mobile manipulation,’ Autonomous Robots, vol. 28, no. 1, pp. 113–130, 2010.
[17] M. Phillips, B. Cohen, S. Chitta and M. Likhachev, ‘E-graphs: Bootstrapping planning with experience graphs,’ in Proceedings of Robotics: Science and Systems, Sydney, Australia, 2012.
[18] S. Koenig, M. Likhachev and D. Furcy, ‘Lifelong planning A*: Artificial Intelligence, vol. 155, no. 1–2, pp. 93–146, 2004.
[19] K. Hauser and J.-C. Latombe, ‘Multi-modal motion planning in non-expansive spaces,’ The International Journal of Robotics Research, vol. 29, no. 7, pp. 897–915, 2010.
[20] T. Lai and P. Morere, ‘Robust Hierarchical Planning with Policy Delegation,’ arXiv preprint arXiv:2010.13033 [cs.AI], 2020. arXiv: 2010.13033 [cs.AI].
[21] T. Lai, ‘Discover Life Skills for Planning as Bandits via Observing and Learning How the Worlds Works,’ in 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2022.
[22] J. D. Gammell, T. D. Barfoot and S. S. Srinivasa, ‘Informed sampling for asymptotically optimal path planning,’ IEEE Transactions on Robotics, vol. 34, no. 4, pp. 966–984, 2018.
[23] T. Lai and F. Ramos, ‘Learning to Plan Optimally with Flow-based Motion Planner,’ arXiv:2010.11123 [cs.RO], 2020. arXiv: 2010.11123 [cs.RO].
[24] B. Ichter, J. Harrison and M. Pavone, ‘Learning sampling distributions for robot motion planning,’ in 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 7087–7094.
[25] R. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. S. Kumar, S. Koenig and H. Choset, ‘PRIMAL: Pathfinding via reinforcement and imitation multi-agent learning,’ IEEE Robotics and Automation Letters, vol. 4, no. 3, pp. 2378–2385, 2019.
[26] T. Lai, W. Zhi, T. Hermans and F. Ramos, ‘Parallelised diffeomorphic sampling-based motion planning,’ in Conference on Robot Learning, PMLR, 2022, pp. 81–90.
[27] T. Lai and F. Ramos, ‘Plannerflows: Learning motion samplers with normalising flows,’ in 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2021, pp. 2542–2548.
[28] D. Berenson, P. Abbeel and K. Goldberg, ‘A robot path planning framework that learns from experience,’ in 2012 IEEE International Conference on Robotics and Automation, IEEE, 2012, pp. 3671–3678.
[29] D. Coleman, I. A. Sucan, M. Moll, K. Okada and N. Correll, ‘Experience-based planning with sparse roadmap spanners,’ in 2015 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2015, pp. 900–905.
[30] S. Klemm, J. Oberländer, A. Hermann, A. Roennau, T. Schamm, J. M. Zollner and R. Dillmann, ‘RRT-Connect: Faster, asymptotically optimal motion planning,’ in 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO), IEEE, 2015, pp. 1670–1677.
[31] T. Lai, ‘Sbp-env: A python package for sampling-based motion planner and samplers,’ Journal of Open Source Software, vol. 6, no. 66, p. 3782, 2021.
[32] Y. Li, Z. Littlefield and K. E. Bekris, ‘Asymptotically optimal sampling-based kinodynamic planning,’ The International Journal of Robotics Research, vol. 35, no. 5, pp. 528–564, 2016.