2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Constant-time Motion Planning with Anytime Refinement for Manipulation

Itamar Mishani*, Hayden Feddock†, Maxim Likhachev*

***Abstract*— Robotic manipulators are essential for future autonomous systems, yet limited trust in their autonomy has confined them to rigid, task-specific systems. The intricate configuration space of manipulators, coupled with the challenges of obstacle avoidance and constraint satisfaction, often makes motion planning the bottleneck for achieving reliable and adaptable autonomy. Recently, a class of constant-time motion planners (CTMP) was introduced. These planners employ a preprocessing phase to compute data structures that enable online planning provably guarantee the ability to generate motion plans, potentially sub-optimal, within a user defined time bound. This framework has been demonstrated to be effective in a number of time-critical tasks. However, robotic systems often have more time allotted for planning than the online portion of CTMP requires, time that can be used to improve the solution. To this end, we propose an anytime refinement approach that works in combination with CTMP algorithms. Our proposed framework, as it operates as a constant time algorithm, rapidly generates an initial solution within a user-defined time threshold. Furthermore, functioning as an anytime algorithm, it iteratively refines the solution's quality within the allocated time budget. This enables our approach to strike a balance between guaranteed fast plan generation and the pursuit of optimization over time. We support our approach by elucidating its analytical properties, showing the convergence of the anytime component towards optimal solutions. Additionally, we provide empirical validation through simulation and real-world demonstrations on a 6 degree-of-freedom robot manipulator, applied to an assembly domain.**

I. Introduction

Manipulation is prevalent across a wide range of applications. It spans from residential to warehouse and factory environments, where tasks often involve a degree of environmental stability and repetition. Recently, a new concept of Constant-Time Motion Planning (CTMP) was introduced, with the aim of guaranteeing the ability to generate a motion plan within a (short) constant time in such environments [1]–[3]. CTMP has proven to be highly valuable in time-critical applications, such as manipulators catching incoming projectiles, operating on conveyor belts, and mail sorting in a mailroom. Although these algorithms frequently yield solutions within mere milliseconds, in numerous domains the allotted time bound for planning extends beyond these limits, thus remaining underutilized.

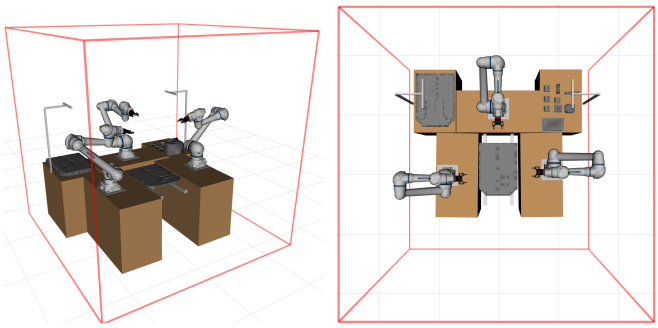


Fig. 1: An assembly cell, composed of three Yaskawa HC10DTP manipulators.

Our inspiration is drawn from manufacturing settings that rely on robotic manipulators for assembly tasks. Common approaches in such settings tend to confine the system to a task-specific configurations, employing a predefined set of motion plans, often designed by humans, thereby constraining autonomy. For example, consider the assembly cell illustrated in Fig. 1, where three robotic arms continuously pick and place objects in a *semi-static* environment [3]. In such an environment, while the general settings remain constant, the manipulated objects can move within their designated regions. This context demands real-time, reliable, and efficient motion planning to instill confidence in autonomous operations. Moreover, as the assembly task involves a sequence of interdependent operations, each building upon previous one, the need for rapid and high quality plan queries becomes crucial.

CTMP approaches employ offline preprocessing to generate auxiliary data structures, which are subsequently utilized in an online query phase to compute paths in constant time. However, they often yield suboptimal solutions. While guaranteeing path generation within milliseconds, the available time budget is frequently longer, offering opportunities for solution improvements. Additionally, in cases of continuous manipulation with successive queries that do not require an arm to return to its home configuration (such as in our assembly example), CTMP typically generates highly suboptimal plans due to its reliance on a pre-defined set of possible start configurations.

Motivated by these challenges, we introduce an anytime variant of these algorithms. This variant employs a preprocessing phase, similar to [1] and [2], that enables the rapid generation of an initial solution — though potentially quite suboptimal — within a constant time constraint. Unlike previous approaches though, the algorithm progressively

I. Mishani and M. Likhachev are with the Robotics Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA. e-mail: {imishani, mlikhach}@.cs.cmu.edu.

H. Feddock is with Electrical and Computer Engineering Department, Pittsburgh University. e-mail: {htf1}@.pitt.edu.

Research was sponsored by the ARM (Advanced Robotics for Manufacturing) Institute through a grant from the Office of the Secretary of Defense and was accomplished under Agreement Number W911NF-17-3-0004. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of the Secretary of Defense or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

improves solution quality while adhering to the defined time budget, which can be determined by user-defined limits or based on previous execution times. In the subsequent sections, we describe the algorithm, provide theoretical analysis of the proposed anytime refinement showing that it converges to an optimal solution if time permits, and demonstrate its utility in the domain of assembly both in simulation and on a physical robot.

II. Related Work

A. Preprocessing-based planning

Preprocessing-based planning techniques are frequently used to offer real-time planning advantages. A common algorithm for doing so is the Probabilistic Roadmaps (PRM), which efficiently preprocesses known environments by generating a dense roadmap, allowing for quick online query responses [4]. Nevertheless, queries might not always connect to the roadmap due to PRM's asymptotic guarantees [5]. Additionally, the connection process relies on the main bottleneck in manipulation planning: collision-detection, which is computationally a very expensive process. Recent work has focused on decomposing the configuration space into a set of collision-free convex sets [6], [7], and subsequently finding smooth trajectories within these sets using optimization methods [8]–[10]. However, since these methods solve an optimization problem over the entire space for each query, they do not provide constant planning time bounds.

An alternative class of preprocessing-based methods rely on past experiences to speed up search [11]–[13]. While speeding up search, none of the aforementioned algorithms can provably ensure fixed planning-time guarantees.

Recently, a planning approach with provable constant-time guarantees (CTMP) has been introduced [1]–[3]. In all its variations, this method relies on a preprocessing phase where a library of intelligently computed paths is utilized during the online phase. In particular, [1] introduces a framework for decomposing a pre-defined region-of-interest (RoI) within a static environment into a set of sub-regions. By computing only one representative path per each sub-region, it enables an online planner to provably guarantee finding a plan to any state within the entire RoI within a user-defined (small) time bound. This concept was subsequently expanded to encompass static environments featuring dynamic goal objects [2], as well as semi-static environments [3]. However, these methods usually generate solutions within a few milliseconds and do not exploit the entire time budget available to them during an online phase. Furthermore, although beneficial for generating fast solutions, these planners operate under the assumption of a predefined set of home configurations (start states) and disregard task sequences. This limitation forces successive plans to traverse the same state, frequently resulting in highly suboptimal paths for manipulation tasks like assembly operations.

B. Anytime Heuristic Search

In scenarios where planning time is restricted, an anytime search algorithm, becomes advantageous, as it aims to rapidly compute an initial, possibly suboptimal solution, and subsequently refine it as time permits. To achieve anytime characteristics in heuristic search, typical approaches involve obtaining fast initial solution through algorithms like beam search, and then progressively increase the number of expandable nodes at a given depth level [14]–[17]. Alternatively, and widely used in robotics, anytime algorithms often incrementally adjust the heuristic inflation [18]–[23]. Nonetheless, none of the above provide planning time bounds for the initial solution.

C. Assembly Planning

The assembly planning problem involves devising a systematic and efficient sequence of actions and motions required to assemble a complex product from individual components. Many such tasks fall under Task and Motion Planning (TAMP), which integrates high-level task planning and low-level motion planning [24], [25]. To address assembly's sequential nature, research often targets optimal sequences [24], [26], [27]. However, typically used motion planners, such as Rapidly-Exploring Random Trees (RRTs) [28]–[30] and PRMs [4], overlook this repetitive and sequential nature. Here, we focus on developing a motion planner that provides completeness and constant-time planning guarantees, using the repetitive nature of the assembly problem for generating fast feasible solutions that are progressively being optimized until the permitted planning time expires..

III. Preliminary

Consider $\mathcal{X}$ as the state space of a robot $\mathcal{R}$ operating within a semi-static environment $\mathcal{W} \subset \mathbb{R}^3$. We assume $\mathcal{R}$ to be controllable and focus on kinematic planning. Consequently, plans are presumed reversible and feasible in both directions. Let $\mathcal{G} \subseteq \mathcal{X}$ denote a region-of-interest (RoI), with $\mathcal{G} = \bigcup_{i=1}^{n} \mathcal{G}_i$ which may potentially consist of a set of disjoint goal regions that we call *local-RoIs* $\mathcal{G}_i$. In an assembly context, these local-RoIs might represent picking or placing regions. Given a goal state $g \in \mathcal{G}$ (e.g., grasping, placing), our objective is to plan the robot's motion to it. We aim to generate these motions quickly within an upper-bound time constraint T_{bound}. However, a trade-off often exists between planning time and path optimality. We note two key observations: Firstly, instances arise where a plan can be computed within a timeframe faster than T_{bound}, affording room to refine the solution using the time difference. Secondly, in domain with sequential nature, the time constraint may prove flexible. If an ongoing path remains incomplete, and there is T_{left} until its execution concludes, the planning time limit can be extended by T_{left}.

We formulate the planning problem as a graph search by discretizing the planning space and representing the RoI (goal region) as a set of states within the graph located in that region.

Previously presented in [1], [2], we re-introduce the notation of *reachable* states.

Definition 1. *Given a planner, a goal state $g \in \mathcal{G}$ is **reachable** from a state s if there exists a path from s to g and it can be computed in finite time.*

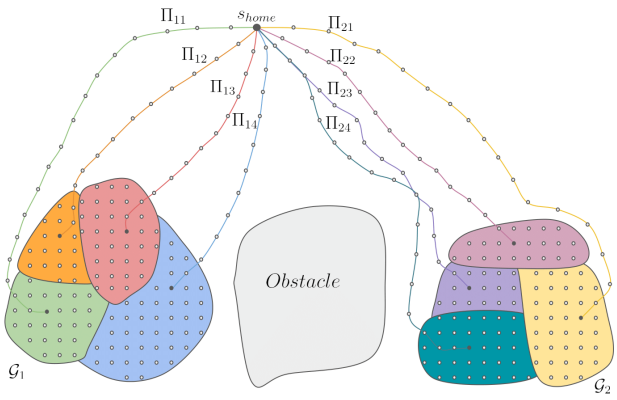


Fig. 2: Illustration of the preprocessing phase. For each local-RoI $\mathcal{G}_i$, we compute neighborhoods, distinguished by different colors, to finally form the cover of $\mathcal{G}$. Moreover, each neighborhood is associated with a path Π_{ij} from s_{home} to an *attractor state*, ensuring that every state within that neighborhood is constant-time feasible from the attractor state.

We wish for our framework to be able to plan a path from any possible state that the robot can be at, to any reachable goal state $g \in \mathcal{G}$ within T_{bound}. Thus, we first define the following:

Definition 2. *We say that a reachable state $g \in \mathcal{G}$ is **constant-time feasible** from a state s if a planner can find a path to it within T_{bound}.*

To achieve this, in an offline phase we decompose our RoI $\mathcal{G}$ into a collection of sets referred to as the *cover*, comprising sub-regions called *neighborhoods*. Formally, we define them as follows:

Definition 3. *A **cover** of $\mathcal{G}$ denoted as $\mathcal{O}$, is a collection of sets o_i each called a **neighborhood** for which:*

- $o_i \subset \mathcal{X}$ *for each* $o_i \in \mathcal{O}$.
- $\mathcal{G} \subseteq \bigcup_{o_i \in \mathcal{O}} o_i$.

Our aim is to use this decomposition to ensure constant-time feasibility for all reachable states within each neighborhood in online settings.

IV. Algorithmic Approach

A. Preprocessing Phase

Our approach to obtaining an initial solution within a constant-time framework is built upon a preprocessing phase that enables efficient online computations. Similarly to [1]–[3], during the offline phase, we assume access to a planner $\mathcal{P}$ and a predefined home state s_{home} (the framework can also be extended to support a finite set of home configurations). A straightforward approach might involve computing and storing paths from s_{home} to all states in $\mathcal{G}$ and all feasible paths between any $s_i, s_j \in \mathcal{G}$ pairs. However, this approach demands excessive memory usage. Alternatively, as depicted in Fig. 2, we can construct neighborhoods around specific states, called *attractor states*, to form a cover $\mathcal{O}$ of $\mathcal{G}$. We construct these neighborhoods such that each state within them can easily reach an attractor state through a path computable within a (small) bounded time. One method, similar to [1], involves creating the neighborhood with a navigation function (e.g., Euclidean distance) such that from any state within the neighborhood, always transitioning to a state that maximally reduces the specified navigation function guarantees a path leading to the attractor state. Another method, akin to [2], guarantees that the path corresponding to a neighborhood's attractor state can be utilized as an experience, enabling constant-time planning to all states within the neighborhood. In this work, we adopt the first method and independently preprocess all $\mathcal{G}_i$, with a shared home state s_{home} as described in Alg. 1.

Algorithm 1: Preprocess

Input : s_{home}: Home state for all local-RoIs
$\mathcal{P}$: Planner
$\mathcal{G}$: RoI
Output: Library $\mathcal{L}$ maps local-RoIs to their cover, containing neighborhoods and their corresponding paths

1 **Procedure** `Preprocess`(s_{home}, $\mathcal{P}$, $\mathcal{G}$)
2 $\mathcal{L} \leftarrow$ `InitHashMap ()` `// A hashmap from a local-RoI to its data`
3 **for** $\mathcal{G}_i$ *in* $\mathcal{G}$ **do**
4 $\mathcal{G}_i^{covered} \leftarrow \emptyset$
5 **while** $\mathcal{G}_i \setminus \mathcal{G}_i^{covered} \neq \emptyset$ **do**
6 $s_{attractor} \leftarrow$ `SampleValidUncovState` ($\mathcal{G}_i$)
7 $\tilde{\Pi} = \mathcal{P}.PlanPaths(s_{attractor}, s)$ `// Compute a set of representative paths`
8 **if** $\tilde{\Pi} = \emptyset$ `// No valid path`
9 **then**
10 continue
11 $o \leftarrow$ `ConstrcutNeighborhood` ($s_{attractor}$)
12 $\mathcal{G}_i^{covered} \leftarrow \mathcal{G}_i^{covered} \bigcup o$
13 $\mathcal{L}[\mathcal{G}_i] \leftarrow \mathcal{L}[\mathcal{G}_i] \bigcup (s_{attractor}, o, \tilde{\Pi})$
14 **return** $\mathcal{L}$

The algorithm iterates through all local-RoIs ($\mathcal{G}_i$), computing the neighborhoods to add to the cover (line 3). For every local-RoI, it samples a valid yet uncovered state (line 6). If a feasible path exists to this state, the algorithm chooses it as attractor state and expands a neighborhood around it (line 11). Adhering to the approach outlined in [1], during the expansion of the neighborhood, we look for all states that can construct a path to the attractor state by always moving to a successor that decreases the navigation function the most. Additionally, the sampling process can be made informed and efficient, rather than being completely random. We continuously track the frontier of the expanded neighborhood during its construction. Once a neighborhood is returned and the sampler is invoked again, the cached frontier set comprises valid states that lie outside the bounds of the previously constructed neighborhood. Subsequently, we attempt to sample the next attractor state from this frontier set (More details can be found in [1]).

B. Online Phase

1) Initial Solution: During the online phase, our aim is to leverage preprocessed data to rapidly generate initial plans within a specified time limit. The core concept is straightforward: utilizing a lookup approach, we can associate the queried point $g \in G$ with its corresponding neighborhood and ascertain the path from s_{home} to $s_{attractor}$ associated with this specific neighborhood followed by connecting g to $s_{attractor}$. The connection from g to $s_{attractor}$ corresponds to following a sequence of states from g that minimize

the navigation function w.r.t. $s_{attractor}$ (e.g., decreasing Euclidean distance w.r.t. $s_{attractor}$). Having the assumption of reversibility, we concatenate the path from s_{home} to $s_{attractor}$ with the reversed path from $s_{attractor}$ to g.

Having access to preprocessed data where all valid states $g \in \mathcal{G}$ are constant-time feasible from s_{home} we claim the following:

Definition 4. *Consider the collection of all states that are constant-time feasible within $\mathcal{G}$ from s_{home}:*

$$\mathcal{F} = \{g \in \mathcal{G} |\ g \text{ is constant-time feasible from } s_{home}\}$$

Furthermore, let ρ denote the set of all states across the precomputed paths:

$$\rho = \{s \in \bigcup_{i=1}^{n} \bigcup_{j} \Pi_{ij}\}$$

*The set of all robot **potential states** is defined as:*

$$\Phi = \mathcal{F} \bigcup \rho$$

Proposition 1. *All reachable goal poses $g \in \mathcal{F}$ are constant-time feasible from any potential state $s \in \Phi$.*

Since both states $s \in \Phi$ and $g \in G$ are constant-time feasible from s_{home}, and considering the reversibility of paths, we can directly concatenate one path with the reverse of the other path.

Algorithm 2: Query

Input : s: start state ($s \in \Phi$)
g: goal state ($g \in \mathcal{G}$)
s_{home}: Home state
T_{bound}: Time budget
$\mathcal{L}$: The preprocessed library
Output: Path Π from start state to goal state

1 **Procedure** `Query`$(s,\ g,\ s_{home},\ T_{bound},\ \mathcal{L})$
2 $T_{start} =$ `GetCurrentTime`()
3 **if** $\Pi_{home,g} \leftarrow$ `FindRepPath` $(g,\ \mathcal{L}) \neq \emptyset$ `// Find the local-RoI, the specific neighborhood and the corresponding representative path`
4 **then**
5 $\Pi_{home,g} \leftarrow$ `Connect`$(\Pi_{home,g}, g)$ `// Connect g to the path (e.g., greedy search)`
6 **if** $s \neq s_{home}$ **then**
7 $\Pi_{home,s} \leftarrow$ `FindRepPath` $(s,\ \mathcal{L})$
8 **if** $\Pi_{home,s} = \emptyset$ **then**
9 **return** $\emptyset$
10 $\Pi_{home,s} \leftarrow$ `Connect`$(\Pi_{home,s}, s)$
11 $\Pi =$ `Reverse`$(\Pi_{home,s}) \cdot \Pi_{home,g}$
`// Concatenate the two paths`
12 **else**
13 $\Pi = \Pi_{home,g}$
14 $\Pi \leftarrow$ `AnytimeRefinement`$(s, g, \Pi, T_{start}, T_{bound})$
15 **return** Π
16 **else**
17 **return** $\emptyset$

2) Solution Refinement: In this section, we present the key contribution of the paper. Given that the initial path is derived from lookups and processes which do not require validity checks, the querying process typically takes up to a few milliseconds (Alg. 2, lines 2-12). Nonetheless, the allocated time budget is frequently longer and can be utilized to improve the initial solution. The initial paths can be suboptimal since they first move towards the attractor state and then proceed toward the actual goal. Additionally, a more pronounced issue arises in the case of two sequential queries: after completing the first query, the arm must navigate all the way back to its home state before it can proceed to its new goal.

Algorithm 3: Anytime Refinement

Input : s_{start}: start state ($s_{start} \in \Phi$)
s_{goal}: goal state ($s_{goal} \in \mathcal{G}$)
Π: Initial path
T_{start}: Start time of the query
T_{bound}: Time budget given
Output: Refined Path Π

1 **Procedure** `AnytimeRefinement`$(s_{start},\ s_{goal},\ \Pi,\ T_{start},\ T_{bound})$
2 CLOSED = INCONS = $\emptyset$
3 OPEN = $\{\Pi\}$ `// Insert all states in` Π `with their corresponding g-value`
4 $C = g(s_{goal})$; $\epsilon = \max_{s \in \Pi}(\frac{C - g(s)}{h(s) + \delta})$ `//` $0 < \delta \ll 1$
5
6 **while** $(\texttt{GetCurrentTime}() - T_{start}) < T_{bound}$ *and* $\epsilon > 1$ **do**
7 **while** $(\texttt{GetCurrentTime}() - T_{start}) < T_{bound}$ *and* *OPEN* $\neq \emptyset$ **do**
8 $s_{min} = \min_{s \in OPEN}(g(s) + \epsilon \cdot h(s))$
9 **if** $s_{min} = s_{goal}$ **then**
10 $\Pi \leftarrow$ `ExtractPath`()
11 break
12 insert s_{min} into CLOSED
13 **for** $s' \in Successors(s_{min})$ **do**
14 **if** s' *was not visited before* **then**
15 $g(s') = \infty$
16 **if** $g(s') > g(s_{min}) + c(s, s')$ **then**
17 $g(s') = g(s_{min}) + c(s, s')$
18 **if** $s' \notin$ *CLOSED* **then**
19 insert s' into OPEN
20 **else**
21 insert s' into INCONS
22 CLOSED $\leftarrow \emptyset$
23 $C \leftarrow g(s_{goal})$
24 $\epsilon \leftarrow \min(\max_{s \in \Pi}(\frac{C - g(s)}{h(s) + \delta}),\ \max_{s \in OPEN}(\frac{C - g(s)}{h(s) + \delta}))$
25 OPEN $\leftarrow$ OPEN $\bigcup$ INCONS $\bigcup \Pi$
26 **return** Π

With an initial path and remaining planning time, our objective is to employ this initial solution as an experience for an anytime refinement search (Alg. 3). As presented in Sec. II-B, one approach to conducting an anytime heuristic search involves progressively reducing the heuristic's inflation. We adopt a methodology similar to [18], with adaptations.

The anytime-repairing A* algorithm (ARA*) operates through a sequence of weighted A* iterations, gradually reducing heuristic inflation. In standard A*, expanded states are ensured to be consistent and thus not re-expanded. However, in weighted A*, due to heuristic inflation, inconsistent heuristics can be employed in practice, resulting in potential multiple expansions for a single state. ARA* addresses this by limiting expansions to once per state and designating states with improved g-values as *locally-inconsistent*, adding them to an INCONS list. At the conclusion of an iteration and upon achieving a corresponding sub-optimal solution, the subsequent iteration's open-list is initialized with previously identified inconsistent states, enabling the reuse of earlier computational efforts. A key limitation of ARA* is that it does not provide a guarantee of finding the initial solution within a specific time bound. Moreover, particularly in scenarios involving high-dimensional state spaces, finding

the initial solution is often non-trivial.

We observe that given the initial solution, and knowing it is likely to be suboptimal, we consider it to be a collection of potentially inconsistent states. Thus, we initiate the open list of the search with all the states on the path along with their current g-values (Alg. 3 line 3). This approach facilitates the potential re-expansion of these specific states, thereby enabling the search to discover improved paths towards them. Nevertheless, this process relies on the heuristic inflation value: the higher it is, the less exploration we undertake, given that the goal state already exists in the open list with a heuristic value of zero. We can leverage this observation and balance between exploration and quick refinements.

As suggested in [31] and demonstrated in [18], the suboptimality of a path can be more tightly bounded. Between successive executions searches, a tighter bound can be established using the current path cost C. Unlike ARA*, where this insight serves solely to report suboptimality, here we exploit the value of C to determine both the initial weight and its subsequent updates (Alg. 3, lines 4 and 24).

Given the current path cost C, which is effectively the g-value of the goal state, we want at least one state s from the open list or the current path such that:

$$g(s) + \epsilon \cdot h(s) < C \tag{1}$$

Which leads us to the following inequality condition

$$\epsilon < \frac{C - g(s)}{h(s)} \tag{2}$$

Inequality 2 signifies that, as the goal state exists within the open list at the beginning of each iteration, in order for it not to be expanded prematurely, the inflation of the heuristic must be less than the above expression. However, as the time available for refinement is determined online, our objective is for each iteration to improve the solution as fast as possible.

Consequently, for the first iteration, we opt to initialize the weight with the maximum attainable value that still guarantees at least one expansion from the initial path (Alg. 3 line 4, where δ assists us in dealing with singularities while ensuring the satisfaction of the inequality condition).

After a search iteration, a new weight must be set for the next iteration, aiming to expand at least one non-goal state. However, re-establishing the weight as in line 4 could risk undermining convergence to an optimal solution. If the path remains unimproved, C and path states persist. Attempting to reset inflation the same way could initiate another search with the same weight, hindering progress. To address this, between iterations, we set the new weight as the minimum between the maximum current path value and the maximum OPEN list value (Alg. 3 line 24).

C. Anytime Refinement: Theoretical guarantees

Given Infinite time budget, we want to show that our algorithm converges to the optimal solution.

Lemma 1. *Assuming a consistent (monotonic) heuristic function $h : \mathcal{X} \to \mathbb{R}$ (which implies admissibility, i.e., it never overestimates the cost to reach the goal), and denoting the optimal solution cost as C^*, we observe that if, at any given iteration, $\epsilon = 1$, then the resultant cost is $C = C^*$.*

Proof. The proof is straightforward. As A* search with a consistent heuristic function is guaranteed to return an optimal solution, as indicated in [18], when the heuristic inflation reaches 1, our algorithm effectively operates as A*, ensuring the attainment of the optimal solution. □

Lemma 2. *The update rule for the inflation of the heuristic*

$$\epsilon = \min(\max_{s\in\Pi}(\frac{C - g(s)}{h(s) + \delta}), \max_{s\in OPEN}(\frac{C - g(s)}{h(s) + \delta})), \quad 0 < \delta \ll 1$$

*is **strictly decreasing***

Proof. Consider ϵ_i as the current heuristic weight, and ϵ_{i+1} as the weight for the next iteration. Furthermore, let C represent the current iteration cost. We know that:

$$g(s) + \epsilon_i \cdot h(s) \geq C, \quad \forall s \in \textit{OPEN}$$

Additionally, let $\delta > 0$. Thus,

$$\epsilon_i > \frac{C - g(s)}{h(s) + \delta}, \quad \forall s \in \textit{OPEN}$$

This leads us to the result:

$$\epsilon_i > \max_{s\in OPEN} \frac{C - g(s)}{h(s) + \delta} \geq$$

$$\geq \min(\max_{s\in\Pi}(\frac{C - g(s)}{h(s) + \delta}), \max_{s\in OPEN}(\frac{C - g(s)}{h(s) + \delta})) = \epsilon_{i+1}$$

□

Theorem 1. *Alg. 3 is guaranteed to converge to the optimal solution.*

Proof. The proof follows directly from Lemma 1 and 2. □

V. Experiments

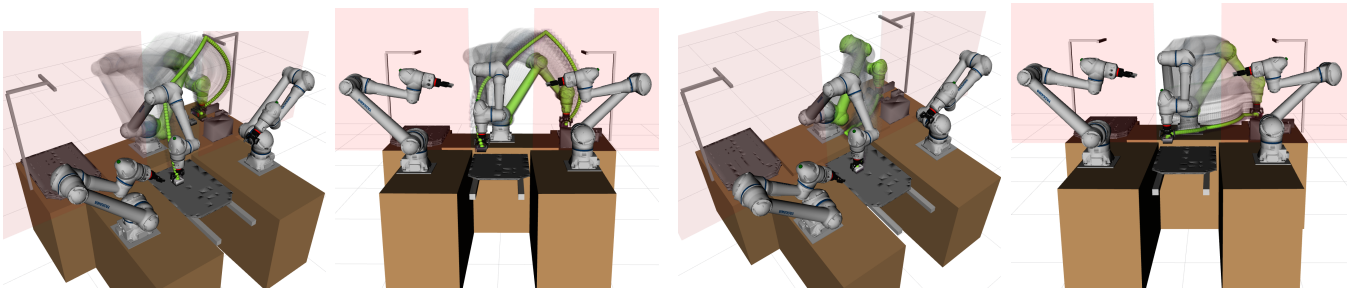
To assess our approach, we conducted an assembly experiment using a 6-DOF robotic arm in simulation and demonstrated its capabilities in both simulated and real-world scenarios[1].

A. Manipulation Implementation: A Case Study on Assembly

Let us have a robotic arm $\mathcal{R}$. Let its configuration space denoted as $\mathcal{C}$ and its task space be $\Gamma \subset SE(3)$. In our assembly task, we consider the general task of pick-and-place. Since we are constantly changing the manipulated regions by removing and placing objects, we define the local-RoIs as the *pre-grasp* and *pre-place* regions in Γ. Nonetheless, a path is being computed in $\mathcal{C}$.

1) State validity check: Considering that neighborhoods are constructed within Γ, the validation process for task space states within $\mathcal{G}$ involves verifying the existence of an inverse kinematics solution and then assessing graspability and placeability (i.e. evaluating the feasibility of grasping and placing operations from the given state). This is accomplished by generating grasp and pose action trajectories through Runge-Kutta integration.

[1]See supplementary material for accompanying videos.



(a) General view: **without** refinement (b) Front view: **without** refinement (c) General view: **with** refinement (d) Front view: **with** refinement

Fig. 3: Comparing CTMP with and without refinement. Fig. 3a and 3b show general and front views of the path from pick pose (green) to place pose (solid gray), without refinement, having to pass through s_{home}. The green line is the end-effector trajectory. Planning time: 38 msec, cost: 82 steps, suboptimallity: 20.4. Fig. 3c and 3d show general and front views with anytime refinement. Here, the path do not pass through s_{home}. Refining time: 1700 msec, cost: 32 steps, suboptimallity: 4.9.

TABLE I: Experimental results comparing our method with 7 different planners

		Our Method	Shortcut-CTMP	CTMP	PRM	RRT*	ARA*	E-Graphs	RRT-Connect
Using s_{home} as start state	Success rate [%]	**100**	100	100	97	84	75	90	96
	Avg. Path Cost [steps]	**16.7**	27.8	44.1	55.3	49.4	22.4	18.3	58.7
	Planning time [msec]	500 ± 0	244 ± 223	17 ± 1	90 ± 5	500 ± 0	500 ± 0	72 ± 52	82 ± 37
Using random potential state $s \in \Phi$ as start state	Success rate [%]	**100**	100	100	98	82	50	72	96
	Avg. Path Cost [steps]	**35.4**	48.1	88.2	66.1	76.0	28.8	75.5	80.3
	Planning time [msec]	2089 ± 624	459 ± 16	37 ± 3	92 ± 1	1857 ± 720	1823 ± 713	247 ± 12	116 ± 1

2) Anytime refinement planning from any potential state to a goal state: As assembly is a sequential process, the planner continuously receives consecutive queries, where any potential state $s \in \Phi$ can serve as the initial state for the planning request. From any potential state, there exists a path to s_{home}. In assembly, the start state can be a previous goal state, a state along a previously queried path, or s_{home}. This enables us to simplify Alg. 2 even more, since lines 7-10 are now trivially computed; $\Pi_{home,s}$ is equal to or a subset of the previous queried path. Still, the concatenated path may result in a highly suboptimal solutions without the refinement approach, as depicted in Figure 3.

3) Anyobject Extension: In manipulation, particularly during assembly tasks, there is a constant interplay between grasping and placing objects which alters the arm's kinematic chain and therefore its collision model. To ensure collision-free solutions, our algorithm is extended to accommodate any-object manipulation. Having a set of object primitives and their maximum allowable sizes (sphere, box, cylinder, etc.), in Algorithm 1, line 7, a set of paths is computed. Initially, we calculate a path without any object attached. Subsequently, for each object primitive, we determine the largest collision-free dimensions that can be accommodated by that path. The trajectory is recomputed with the colliding dimensions and this process is repeated until the object reaches the maximum allowable size. We repeat these steps for each object, culminating in a set of object-path pairs which we can query online given the object primitive and size.

B. Evaluation

In this experimental setting, we preprocessed 4 local-RoIs, allowing primitives in the x, y, and z dimensions, along with a single rotational degree of freedom (roll, pitch, or yaw). The preprocessing stage (Alg. 1) took 126 minutes and used 88 MB of memory for the entire RoI. We compared our method with seven algorithms: CTMP without anytime refinement, CTMP with shortcutting, PRM, RRT*, ARA*, E-Graphs and RRT-Connect, presented in Table I. All experimental results were run on Intel Core i9-12900H with 64GB RAM (4.7GHz). The top three rows provide statistics for 100 queries using the home state (s_{home}) as the initial state. A timeout of 500 milliseconds was set for all planners. The bottom 3 rows provide statistics for 50 queries between random potential states ($s_{start} \in \Phi$) from a pick region as initial states to random goal states in place region. Here, we randomly assigned extra planning time from 500 msec to 3 sec, mirroring real scenarios where planners query during ongoing execution. The ARA* search was initiated with a weight of 50, which was then gradually decreased by 5. For the E-graphs approach, we utilized 8 experiences from the CTMP preprocess phase, selecting 2 experiences from each local-RoI. Each of these experiences represented a path leading to a central state within a randomly chosen neighborhood. In both experiments, while maintaining a 100% success rate, our planner also notably enhanced the solution quality. The results demonstrate that our planner not only ensures provably constant-time completeness but also generates quality solutions when the allocated time budget permits.

VI. Conclusions

We have introduced an anytime adaptation of CTMP, demonstrating its capability to converge towards optimal solutions. Its applicability in manipulation tasks was showcased in the context of assembly's pick-and-place operations. Future research could leverage the framework of CTMP and utilize its solutions as experiences for making real-time adjustments to minor environment changes.

REFERENCES

[1] F. Islam, O. Salzman, and M. Likhachev, "Provable indefinite-horizon real-time planning for repetitive tasks," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, no. 1, pp. 716–724, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/ICAPS/article/view/3540

[2] F. Islam, O. Salzman, A. Agarwal, and M. Likhachev, "Provably constant-time planning and replanning for real-time grasping objects off a conveyor belt." [Online]. Available: http://arxiv.org/abs/2003.08517

[3] F. Islam, C. Paxton, C. Eppner, B. Peele, M. Likhachev, and D. Fox, "Alternative paths planner (APP) for provably fixed-time manipulation planning in semi-structured environments." [Online]. Available: http://arxiv.org/abs/2012.14970

[4] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[5] L. Kavraki, M. Kolountzakis, and J.-C. Latombe, "Analysis of probabilistic roadmaps for path planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 1, pp. 166–171, 1998.

[6] H. Dai, A. Amice, P. Werner, A. Zhang, and R. Tedrake, "Certified polyhedral decompositions of collision-free configuration space," 2023.

[7] M. Petersen and R. Tedrake, "Growing convex collision-free regions in configuration space using nonlinear programming," 2023.

[8] T. Marcucci, J. Umenberger, P. A. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," 2023.

[9] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," 2022.

[10] T. Cohn, M. Petersen, M. Simchowitz, and R. Tedrake, "Non-euclidean motion planning with graphs of geodesically-convex sets," 2023.

[11] 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*, 2012, pp. 3671–3678.

[12] D. Coleman, I. A. Şucan, M. Moll, K. Okada, and N. Correll, "Experience-based planning with sparse roadmap spanners," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 900–905.

[13] 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, July 2012.

[14] W. Zhang, "Complete anytime beam search," in *AAAI/IAAI*, 1998, pp. 425–430.

[15] S. G. Vadlamudi, S. Aine, and P. Chakrabarti, "Incremental beam search," *Information Processing Letters*, vol. 113, no. 22-24, pp. 888–893, 2013.

[16] S. Lemons, W. Ruml, C. L. López, and R. Holte, "Triangle search: An anytime beam search," in *ICAPS 2023 Heuristics and Search for Domain-Independent Planning Workshop*, 2023.

[17] S. Aine, P. Chakrabarti, and R. Kumar, "Awa-a window constrained anytime heuristic search algorithm," in *International Joint Conference on Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 2007, pp. 2250–2255.

[18] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643, 2008. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S000437020800060X

[19] E. A. Hansen and R. Zhou, "Anytime heuristic search," *Journal of Artificial Intelligence Research*, vol. 28, pp. 267–297, 2007.

[20] L. Cohen, M. Greco, H. Ma, C. Hernández, A. Felner, T. S. Kumar, and S. Koenig, "Anytime focal search with applications." in *IJCAI*, 2018, pp. 1434–1441.

[21] J. van den Berg, R. Shah, A. Huang, and K. Goldberg, "Anytime nonparametric a*," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, no. 1, pp. 105–111, Aug. 2011. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/7819

[22] M. Phillips, A. Dornbush, S. Chitta, and M. Likhachev, "Anytime incremental planning with e-graphs," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 2444–2451.

[23] R. Natarajan, M. Saleem, S. Aine, M. Likhachev, and H. Choset, "A-mha*: anytime multi-heuristic a," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 10, no. 1, 2019, pp. 192–193.

[24] 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, 2023.

[25] 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, no. 1, pp. 265–293, 2021. [Online]. Available: https://doi.org/10.1146/annurev-control-091420-084139

[26] W. Wan and K. Harada, "Integrated assembly and motion planning using regrasp graphs," *Robotics and biomimetics*, vol. 3, no. 1, pp. 1–11, 2016.

[27] V. N. Hartmann, O. S. Oguz, D. Driess, M. Toussaint, and A. Menges, "Robust task and motion planning for long-horizon architectural construction planning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6886–6893.

[28] S. LAVALLE, "Rapidly-exploring random trees : a new tool for path planning," *Research Report 9811*, 1998. [Online]. Available: https://cir.nii.ac.jp/crid/1573950399665672960

[29] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning." [Online]. Available: http://arxiv.org/abs/1105.1186

[30] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2. IEEE, 2000, pp. 995–1001. [Online]. Available: http://ieeexplore.ieee.org/document/844730/

[31] R. Zhou and E. A. Hansen, "Multiple sequence alignment using anytime a*." in *AAAI/IAAI*, 2002, pp. 975–977.