Provable Infinite-Horizon Real-Time Planning for Repetitive Tasks

Fahad Islam, Oren Salzman and Maxim Likhachev
The Robotics Institute, Carnegie Mellon University
{fi,osalzman}@andrew.cmu.edu, maxim@cs.cmu.edu

***Abstract*— In manufacturing, robots often have to perform highly-repetitive manipulation tasks in structured environments. In this work we are interested in the settings where the tasks are similar, yet not identical (e.g., due to uncertain orientation of objects) and motion planning needs to be extremely fast. Preprocessing-based approaches prove to be very beneficial in these settings—they analyze the configuration-space offline to generate some auxiliary information which can then be used in the query phase to speedup planning times. Typically, the tighter the requirement is on query times the larger the memory footprint will be. In particular, for high-dimensional spaces, providing real-time planning capabilities is impractical. Moreover, as far as we are aware of, none of the general-purpose algorithms come with *provable* guarantees on the real-time performance. To this end, we propose a preprocessing-based method that provides provable bounds on the query time while incurring only a small amount of memory overhead in the query phase. We evaluate our method on a 7-DOF robot arm and show a speedup of over tenfold in query time when compared to the PRM algorithm while guaranteeing a maximum query time of less than 4 milliseconds.**

I. Introduction

Consider the problem of a robot picking up objects from a high-speed conveyor belt and placing them into bins (see Fig. 1). Similarly, consider a robot given the task of stock replenishment—moving in a supermarket and loading items from a cart it carries to half-empty shelves. These problems are examples of repetitive tasks performed by (possibly highly articulated) robots in static environments where the start and goal of each repetitive task is similar, yet not identical, to previous tasks. Difference in the exact start and goal position may be due to uncertainty in the environment (objects placed on different parts of the conveyor belt or in different orientation) or due to highly-similar tasks (objects placed in similar positions on a shelf).

As the set of possible start and goal locations may be large, caching pre-computed paths for all these queries in advance is unmanageable. Clearly, once a task is presented to the robot, it can compute a desired path online. However, this may incur large online planning times that may be unacceptable in many settings. For example, in our conveyor-belt setting, reducing the planning time immediately corresponds to faster unloading capabilities. Moreover, if the planner cannot *guarantee* to pick items from the conveyor in a timely manner, the system is required to account for missed items by e.g., additional conveyor belts that will redirect items back to the robot—a costly backup in terms of both time and space. Thus, a natural approach is to preprocess the environment in an offline phase to allow for fast planning times online.

Fig. 1: Motivating scenario—a robot (PR2) picking up objects from a conveyor belt.

One way to preprocess the environment is using the PRM algorithm [4]. Once a a dense roadmap has been precomputed, any query can be efficiently answered online by connecting the source and goal to the roadmap. Query times can be significantly sped up by further preprocessing the roadmaps using landmarks [11]. Unfortunately, there is no guarantee that a query can be connected to the roadmap as PRM only provides *asymptotic* guarantees [3]. Furthermore, this connecting phase requires running a collision-detection algorithm which is typically considered the computational bottleneck in many motion-planning algorithms [8].

Recently, Lehner and Albu-Schäffer [9] suggest the repetition roadmap to extend the PRM for the case of multiple highly-similar scenarios. While their approach exhibits significant speedup in computation time, it still suffers from the previously-mentioned shortcomings.

A complementary approach to aggressively preprocess a given scenario is by minimizing collision-detection time. However this requires designing robot-specific circuitry [10] or limiting the approach to standard manipulators [14].

An alternative approach is to precompute a set of complete paths into a library and given a query, attempt to match complete paths from the library to the new query [1], [2]. Unfortunately, this approach also cannot provide any of the guarantees required by our applications.

Our work bares resemblance to previous work on subgoal graphs [12], [13] and to real-time planning [5], [6], [7]. However, in the former, the entire configuration space is preprocessed in order to efficiently answer queries between *any* pair of states which deems it applicable only to low-dimensional spaces. Similarly, in the latter, to provide guarantees on planning time the search only looks at a finite horizon and interleaves planning and execution.

Our key insight is to combine a precomputed library $\mathcal{L}$ of paths between *several* start and goal configurations together with a method to connect *any* start and goal configuration to a path in $\mathcal{L}$ *without* having to perform collision detection.

This insight allows us to provide *provable bounds* on the time to solve motion-planning queries which is in the order of milliseconds for a seven DOF manipulator.

We evaluate our approach in simulation on the PR2 robot[1] (see Fig. 1) and demonstrate a speedup of over tenfold in query time when compared to the PRM algorithm with a little memory footprint if 0.2 Mbs and while guaranteeing a maximal query time of less than 4 milliseconds.

II. Algorithm Framework

A. Problem Formulation and assumptions

Let $\mathcal{X}$ be the configuration space of a robot operating in a static environment. We are given in advance a start configuration $s_{\text{start}} \in \mathcal{X}$ and some goal region $G \subset \mathcal{X}$. In the query phase we are given multiple queries $(s_{\text{start}}, s_{\text{goal}})$ where $s_{\text{goal}} \in G$ and for each query, we need to compute a collision-free path connecting s_{start} to s_{goal}.

We discretize $\mathcal{X}$ into a state lattice $\mathcal{S}$ such that any state $s \in \mathcal{S}$ is connected to a set of successors via a mapping Succs: $\mathcal{S} \rightarrow 2^{\mathcal{S}}$ and set $G_S := S \cap G$ to be the states that reside in the goal region. We make the following assumptions:

- A1 G_S is a relatively small subset of S. Namely, it is feasible to exhaustively iterate over all states in G_S. However, storing a path from s_{start} to each state in G_S is infeasible.
- A2 The planner has access to a heuristic function $h : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$ which can estimate the distance between any two states in G_S. Moreover the heuristic function should be *weakly-monotonic*, meaning that $\forall s_1, s_2 \in G_S$ where $s_1 \neq s_2 \neq s_{goal}$, it holds that,
$$h(s_1, s_2) \geq \min_{s_1' \in \text{Succs}(s_1)} h(s_1', s_2).$$
Namely, for any distinct pair of states (s_1, s_2) in G_S, at least one of s_1's successors (also belonging to G_S) must have a heuristic value less than or equal to its heuristic value. Note that this assumption does not imply that G is entirely collision free.

These assumptions allow us to establish strong theoretical properties regarding the efficiency of our planner. Namely, that within a known bounded time, we can compute a collision-free path from s_{start} to any state in G_S. Proofs are omitted due to lack of space.

B. Key Idea

Our planner comprises of a preprocessing and a query phase. In the preprocessing phase, G_S is decomposed into two finite sets of (possibly overlapping) subregions $\mathcal{R}$ and $\hat{\mathcal{R}}$. Subregions in $\hat{\mathcal{R}}$ only contain states that are in collision. Each subregion $R_i \in \mathcal{R}$ is a hyper-ball defined using a center which we refer to as the "attractor state" $s_i^{\text{attractor}}$ and a radius r_i. These regions are constructed in such a way that the following two properties hold

- P1 For any goal state $s_{\text{goal}} \in R_i \cap G_S$, a greedy search with respect to $h(s, s_i^{\text{attractor}})$ over $\mathcal{S}$ starting at s_{start} will result in a collision-free path to $s_i^{\text{attractor}}$.

[1]http://www.willowgarage.com/pages/pr2/overview

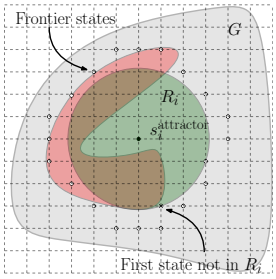

Fig. 2: Visualization of Alg 2. Subregion R_i (green) grown from $s_i^{\text{attractor}}$ in a goal region G_S (grey) containing an obstacle (red). Frontier states and first state not in R_i are depicted by circles and a cross, respectively.

- P2 The union of all the subregions completely cover G_S. Namely, $\forall s \in G_S, \exists R \in \mathcal{R} \cup \hat{\mathcal{R}}$ $s.t.$ $s \in R$.

In the preprocessing stage, we precompute a library of collision-free paths $\mathcal{L}$ which includes one path from s_{start} to each attractor state. In the query phase, given a query s_{goal}, we (i) identify a region R_i such $s_{\text{goal}} \in R_i$ (using the precomputed radii r_i), (ii) run a greedy search towards $s_i^{\text{attractor}}$ by greedily choosing at every point the successor that minimizes h and (iii) append this path with the precomputed path in $\mathcal{L}$ to s_{start} to obtain the complete plan.

C. Algorithm

1) Preprocessing Phase: The preprocessing phase of our algorithm, detailed in Alg. 1, takes as input the start state s_{start} and the goal region G_S and outputs a set of subregions $\mathcal{R}$ and the corresponding library of paths $\mathcal{L}$ from each $s_i^{\text{attractor}}$ to s_{start}.

The algorithm covers G_S by iteratively finding a state s not covered by any region and computing a new region centered at s. To ensure that G_S is complete covered (property P2) we maintain a set V of valid (collision free) and a set I of invalid (in collision) states called *frontier states* (lines 3 and 4, respectively). We start by initializing V with some random state in G_S and iterate until both V and I are empty, which will ensure that G_S is indeed covered.

At every iteration, we pop a state from V (line 8), and if there is no region covering it, we add it as a new attractor state and compute a path π_i to s_{start} (line 11). We then compute the corresponding region (line 12 and Alg. 2).

As we will see shortly, computing a region corresponds to a Dijkstra-like search centered at the attractor state. The search terminates with the region's radius r_i and a list of frontier states that comprise of the region's boundary. The valid and invalid states are then added to V and I, respectively (lines 13 and 14).

Once V gets empty the algorithm starts to search for states which are valid and yet uncovered by growing regions around the states popped from I (lines 16-20). If a valid and uncovered state is found, it is added to V and the algorithm goes back to computing subregions (lines 21-23), otherwise if I also gets empty, the algorithm terminates and it is guaranteed that each valid state contained in G_S is covered under at least one subregion.

Reachability Search: The core of our planner lies in the way we compute the subregions (Alg. 2 and Fig. 2) which

Algorithm 1 Goal Region Preprocessing

Inputs: G_S, s_{start} ▷ goal region and start state
Outputs: $\mathcal{R}, \mathcal{L}$ ▷ subregions and corresponding paths to s_{start}

1: **procedure** PREPROCESSREGION(G_S)
2: $s \leftarrow$ SAMPLEVALIDSTATE(G_S)
3: $V \leftarrow \{s\}$ ▷ valid frontier states initialized to a random state
4: $I = \emptyset$ ▷ invalid frontier states
5: $i \leftarrow 0 \quad \mathcal{L} = \emptyset \quad \mathcal{R} = \emptyset \quad \hat{\mathcal{R}} = \emptyset$
6: **while** V and I are not empty **do**
7: **while** V is not empty **do**
8: $s \leftarrow V$.pop()
9: **if** $\nexists R \in \mathcal{R}$ s.t. $s \in R$ **then** ▷ s is not covered
10: $s_i^{\text{attractor}} \leftarrow s$
11: π_i = PLANPATH($s_i^{\text{attractor}}, s_{\text{start}}$); $\mathcal{L} \leftarrow \mathcal{L} \cup \{\pi_i\}$
12: $(\text{OPEN}, r_i) \leftarrow$ COMPUTEREACHABILITY($s_i^{\text{attractor}}$)
13: insert Valid(OPEN) in V
14: insert Invalid(OPEN) in I
15: $R_i \leftarrow (s_i^{\text{attractor}}, r_i)$; $i \leftarrow i+1$ $\mathcal{R} \leftarrow \mathcal{R} \cup \{R_i\}$
16: **while** I is not empty **do**
17: $s \leftarrow I.pop()$
18: **if** $\nexists R \in \mathcal{R} \cup \hat{\mathcal{R}}$ s.t. $s \in R$ **then** ▷ s is not covered
19: $(X, r) \leftarrow$ SEARCHVALIDUNCOVEREDSTATES(s)
20: $\hat{R} \leftarrow (s, r)$; $\hat{\mathcal{R}} \leftarrow \hat{\mathcal{R}} \cup \{\hat{R}\}$ ▷ invalid region
21: **if** X is not empty **then** ▷ no valid state found
22: insert X in V
23: **break**
24: **return** $\mathcal{R}, \mathcal{L}$

we call a "Reachability Search". The algorithm maintains a set of *reachable* states $S_{\text{reachable}}$ for which property P1 holds. Namely, the greedy successor[2] s' of every reachable state $s \in S_{\text{reachable}}$ is also a reachable state, except for the attractor state $s_i^{\text{attractor}}$. This will ensure that in the query phase, we can run a greedy search from any reachable state $s \in S_{\text{reachable}}$ and it will terminate in the attractor state.

The algorithm computes a subregion that covers the maximum number of reachable states that can fit into a hyperball defined by $h(s, s_i^{\text{attractor}})$. The search maintains a priority queue OPEN ordered according to $h(s, s_i^{\text{attractor}})$. Initially, the predecessors of $s_i^{\text{attractor}}$ are inserted in the OPEN (line 3). For each expanded predecessor, if its valid greedy successor is in $S_{\text{reachable}}$, then the predecessor is also labeled as reachable (lines 10 and 11).

The algorithm terminates when the search pops a state which is valid but does not have a successor state in $S_{\text{reachable}}$ (line 12). Intuitively, this corresponds to the condition when the reachability search exists an obstacle (see Fig. 2). At termination, all the states within the boundary of radius r_i (excluding the boundary) are reachable.

2) Query Phase: For any query goal state $s_{\text{goal}} \in G_S$, we find a subregion $R_i \in \mathcal{R}$ which covers it. Namely, a region R_i for which $h(s_{\text{goal}}, s_i^{\text{attractor}}) < r_i$. We then run a greedy search starting from s_{goal} by iteratively finding for each state s the successor with the minimum heuristic $h(s, s_i^{\text{attractor}})$ value until the search reaches $s_i^{\text{attractor}}$. The traced path is then stitched to the corresponding precomputed path $\pi_i \in \mathcal{L}$. Note that at no point do we need to perform collision checking in the query phase.

[2]A state $s' \in \text{Succ}(s)$ is said to be a *greedy* successor according to some heuristic $h(\cdot)$ if it has the minimal h-value among all successors.

Algorithm 2 Reachability Search

1: **procedure** COMPUTEREACHABILITY($s_i^{\text{attractor}}$)
2: $S_{\text{reachable}} \leftarrow \{s_i^{\text{attractor}}\}$ ▷ Reachable set
3: OPEN $\leftarrow \{\text{Preds}(s_i^{\text{attractor}})\}$ ▷ key: $h(s, s_i^{\text{attractor}})$
4: CLOSED $\leftarrow \emptyset$
5: $r_i \leftarrow 0$
6: **loop**
7: $s \leftarrow$ OPEN.pop()
8: insert s in CLOSED
9: $s'_g \leftarrow \arg\min_{s' \in \text{Succ}(s)} h(s', s_i^{\text{attractor}})$ ▷ greedy succesor
10: **if** $s'_g \in S_{\text{reachable}}$ and Valid(edge(s,s'_g)) **then**
11: $S_{\text{reachable}} \leftarrow S_{\text{reachable}} \cup \{s\}$ ▷ s is greedy
12: **else if** Valid(s) **then**
13: **return** r_i
14: $r_i \leftarrow h(s, s_i^{\text{attractor}})$
15: **for** each $p \in \text{Preds}(s)$ **do**
16: **if** $p \notin$ CLOSED **then**
17: insert p in OPEN with priority $h(p, s_i^{\text{attractor}})$

	PRM (T)	PRM (2T)	PRM (4T)	Our method
Planning time [ms]	6.3 (11.6)	6.6 (11.6)	8.6 (13.0)	0.48 (1.01)
Success rate [%]	92	97	100	100

TABLE I: Experimental results comparing our method with PRM. The table shows the mean/worst-case planning times and success rates for our method and for PRM preprocessed with equal, double and quadruple the time that our method takes in precomputation (T = 984 seconds).

The query time comprises of (i) finding the containing subregion R_i and (ii) running the greedy search to $s_i^{\text{attractor}}$. Step (i) requires iterating over all subregions (in the worst case) which takes $O(|\mathcal{R}|)$ steps while step (ii) requires expanding a number of states which is proportional to r_i. Thus, after the preprocessing stage the maximal query time can be computed.

III. EVALUATION

We evaluated our algorithm by getting some preliminary results on the PR2 robot for the single-arm (7 DOF) planning problem. The task here is to repeatedly pick up objects from a conveyor belt and put them in a bin. We define the task-relevant goal region G by bounding the position and orientation of the end effector.

We compared our approach with the PRM algorithm in terms of success rate and planning times (see Table I) for 100 uniformly sampled goal states from G. Preprocessing (Alg. 1) took 984 seconds and returned 1,865 subregions. For a fair comparision, for PRM the paths from all the nodes in G to s_{start} were precomputed. In the query phase if PRM's connect operation fails for a given query, we consider it a failure. We also bootstrap PRM with 20 goal states from G. Note that the worst-case time for our method shown in these results ($\sim$1 millisecond) is the empirical one and not the computed provable time bound which is 4 milliseconds for this environment.

IV. CONCLUSION

We proposed a preprocessing-based motion planning algorithm that provides provable real-time performance guarantees for repetitive tasks and showed preliminary results. We aim to perform experiments on the hardware with the conveyor setup to demonstrate the efficiency of our planner on a real world system. Moreover on the theoretical side, we aim to provide guarantees on the solution quality.

REFERENCES

[1] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A robot path planning framework that learns from experience. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3671–3678, 2012.

[2] Nikolay Jetchev and Marc Toussaint. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Autonomous Robots*, 34(1-2):111–127, 2013.

[3] Lydia E. Kavraki, Mihail N. Kolountzakis, and Jean-Claude Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Trans. Robotics and Automation*, 14(1):166–171, 1998.

[4] Lydia E Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12(4):566–580, 1996.

[5] Sven Koenig and Maxim Likhachev. Real-time adaptive A*. In *International joint conference on Autonomous agents and multiagent systems*, pages 281–288. ACM, 2006.

[6] Sven Koenig and Xiaoxun Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems*, 18(3):313–341, 2009.

[7] Richard E Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.

[8] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[9] Peter Lehner and Alin Albu-Schaffer. The Repetition Roadmap for Repetitive Constrained Motion Planning. *IEEE Robotics and Automation Letters*, 2018. to appear.

[10] Sean Murray, Will Floyd-Jones, Ying Qi, Daniel J. Sorin, and George Konidaris. Robot Motion Planning on a Chip. In *Robotics: Science and Systems (RSS)*, 2016.

[11] Brian Paden, Yannik Nager, and Emilio Frazzoli. Landmark guided probabilistic roadmap queries. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4828–4834, 2017.

[12] Tansel Uras and Sven Koenig. Feasibility Study: Subgoal Graphs on State Lattices. In *Symposium on Combinatorial Search, SOCS*, pages 100–108, 2017.

[13] Tansel Uras and Sven Koenig. Fast Near-Optimal Path Planning on State Lattices with Subgoal Graphs. In *Symposium on Combinatorial Search, SOCS*, pages 106–114, 2018.

[14] Yiming Yang, Wolfgang Merkt, Vladimir Ivan, Zhibin Li, and Sethu Vijayakumar. HDRM: A Resolution Complete Dynamic Roadmap for Real-Time Motion Planning in Complex Scenes. *IEEE Robotics and Automation Letters*, 3(1):551–558, 2018.