Provable Infinite-Horizon Real-Time Planning for Repetitive Tasks

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.

1 Introduction

We consider the problem of planning robot motions for highly-repetitive tasks while ensuring bounds on the planning times. As a running example, consider the problem of a robot picking up objects from a high-speed conveyor belt and placing them into bins (see Fig. 1). While the environment does not change, 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 as objects may be placed on different parts of the conveyor belt or in different orientation. Fast planning times immediately correspond 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.

Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

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

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. Alternatively, we could attempt to precompute for each start and goal pair a robot path. However, as the set of possible start and goal locations may be large, caching precomputed paths for all these queries in advance is unmanageable in high-dimensional configuration spaces. Thus, we need to balance memory constraints while providing provable real-time query times.

As we detail in Sec. 2, there has been intensive work for fast online planning (Lehner and Albu-Schaffer 2018) and for learning from experience in known environments (Phillips et al. 2012; Phillips et al. 2013; Berenson, Abbeel, and Goldberg 2012b; Coleman et al. 2015). Similarly, compressing precomputed data structures in the context of motion-planning is a well-studied problem with efficient algorithms (Salzman et al. 2014; Dobson and Bekris 2014). However, to the extent of our knowledge, there is no approach that can *provably guarantee* that a solution will be found to *any* query with bounds on planning time and using a low memory footprint.

Our key insight is that given any state s, we can efficiently compute a set of states for which a greedy search towards s is collision free. Importantly, the runtime-complexity of such a greedy search is bounded and there is no need to perform computationally-complex collision-detection operations. This insight allows us to generate offline a small set of so-called "attractor vertices" together with a path between each attractor state and the goal. In the query phase, a path is generated by performing a greedy search to an attractor state followed by the precomputed path to the goal.

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.

2 Related work

One way to preprocess the environment is using the PRM algorithm (Kavraki et al. 1996). Once a a dense roadmap has been pre-computed, 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 (Paden, Nager, and Frazzoli 2017). Unfortunately, there is no guarantee that a query can be connected to the roadmap as PRM only provides *asymptotic* guarantees (Kavraki, Kolountzakis, and Latombe 1998). Furthermore, this connecting phase requires running a collision-detection algorithm which is typically considered the computational bottleneck in many motion-planning algorithms (LaValle 2006).

Recently, Lehner and Albu-Schäffer (Lehner and Albu-Schaffer 2018) 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 (Murray et al. 2016) or limiting the approach to standard manipulators (Yang et al. 2018).

An alternative approach to address our problem 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 (Berenson, Abbeel, and Goldberg 2012a; Jetchev and Toussaint 2013). Using paths from previous search episodes (also known as using experience) has also been an active line of work (Phillips et al. 2012; Phillips et al. 2013; Berenson, Abbeel, and Goldberg 2012b; Coleman et al. 2015). Some of these methods have been integrated with sparse motion-planning roadmaps (see e.g., (Salzman et al. 2014; Dobson and Bekris 2014)) to reduce the memory footprint of the algorithm. Unfortunately, non of the mentioned algorithms provide any of the guarantees required by our applications.

Our work bares resemblance to previous work on subgoal graphs (Uras and Koenig 2017; Uras and Koenig 2018) and to real-time planning (Koenig and Likhachev 2006; Koenig and Sun 2009; Korf 1990). 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. Also their *connect* step is expensive as it requires a Dijkstra's like search to connect to the subgoal graph. Similarly, in the latter, to provide guarantees on planning time the search only looks at a finite horizon and interleaves planning and execution.

Finally, our notion of attractor states bares resemblance to control-based methods that ensure safe operation over local regions of the free configuration space (Conner, Rizzi, and Choset 2003; Conner, Choset, and Rizzi 2006). These regions are then used within a high-level motion planner to compute collision-free paths.

3 Algorithm Framework

3.1 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: $\mathcal{S} \to 2^{\mathcal{S}}$ and set $G_{\mathcal{S}} := \mathcal{S} \cap G$ to be the states that reside in the goal region. We make the following assumptions:

A1 $G_{\mathcal{S}}$ is a relatively small subset of S. Namely, it is feasible to exhaustively iterate over all states in $G_{\mathcal{S}}$. However, storing a path from s_{start} to each state in $G_{\mathcal{S}}$ is infeasible.

A2 The planner has access to a heuristic function $h : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ which can estimate the distance between any two states in $G_{\mathcal{S}}$. Moreover the heuristic function should be *weakly-monotonic*, meaning that $\forall s_1, s_2 \in G_{\mathcal{S}}$ 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 $G_{\mathcal{S}}$, at least one of s_1's successors (also belonging to $G_{\mathcal{S}}$) 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 $G_{\mathcal{S}}$. Proofs are omitted due to lack of space.

3.2 Key Idea

Add a figure that demonstrates the complete approach.

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

Our planner comprises of a preprocessing and a query phase. In the preprocessing phase, $G_{\mathcal{S}}$ 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 $s_{\text{goal}} \in R_i \cap G_{\mathcal{S}}$, a greedy search with respect to $h(s, s_i^{\text{attractor}})$ over $\mathcal{S}$ starting at s_{goal} will result in a collision-free path to $s_i^{\text{attractor}}$.

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

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.

3.3 Algorithm

Preprocessing Phase The preprocessing phase of our algorithm, detailed in Alg. 1, takes as input the start state s_{start} and the goal region $G_{\mathcal{S}}$ 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 $G_{\mathcal{S}}$ by iteratively finding a state s not covered by any region and computing a new region centered at s. To ensure that $G_{\mathcal{S}}$ 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 $G_{\mathcal{S}}$ and iterate until both V and I are empty, which will ensure that $G_{\mathcal{S}}$ 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 $G_{\mathcal{S}}$ is covered under

Algorithm 1 Goal Region Preprocessing

```
   Inputs: $G_{\mathcal{S}}$, $s_{\text{start}}$                  ▷ goal region and start state
   Outputs: $\mathcal{R}, \mathcal{L}$   ▷ subregions and corresponding paths to $s_{\text{start}}$
1: procedure PREPROCESSREGION($G_{\mathcal{S}}$)
2:     $s \leftarrow$ SAMPLEVALIDSTATE($G_{\mathcal{S}}$)
3:     $V \leftarrow \{s\}$        ▷ valid frontier states initialized to a random state
4:     $I = \emptyset$                              ▷ invalid frontier states
5:     $i \leftarrow 0$   $\mathcal{L} = \emptyset$   $\mathcal{R} = \emptyset$   $\hat{\mathcal{R}} = \emptyset$
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:                $\pi_i$ = PLANPATH($s_i^{\text{attractor}}$, $s_{\text{start}}$);   $\mathcal{L} \leftarrow \mathcal{L} \cup \{\pi_i\}$
12:                (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 $\mathcal{R}, \mathcal{L}$
```

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 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 $s \in S_{\text{reachable}}$

[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.

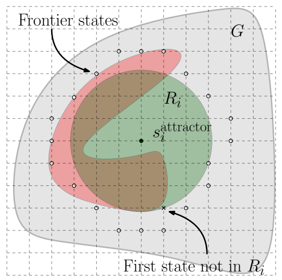

Figure 2: Visualization of Alg 2. Subregion R_i (green) grown from $s_i^{\text{attractor}}$ in a goal region $G_{\mathcal{S}}$ (grey) containing an obstacle (red). Frontier states and first state not in R_i are depicted by circles and a cross, respectively.

Algorithm 2 Reachability Search

```
1:  procedure ComputeReachability($s_i^{\text{attractor}}$)
2:      $S_{\text{reachable}} \leftarrow \{s_i^{\text{attractor}}\}$        ▷ Reachable set
3:      OPEN $\leftarrow$ {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 $p \in$ Preds($s$) do
16:             if $p \notin$ CLOSED then
17:                 insert $p$ in OPEN with priority $h(p, s_i^{\text{attractor}})$
```

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 greedy 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.

Query Phase For any query goal state $s_{\text{goal}} \in G_{\mathcal{S}}$, 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.

4 Analysis

This is wrong unless we talk about tie breaking. I don't think that tie breaking is an implementation detail and we should mention it in the prev. section

Lemma 1. *A greedy search with respect to $h(s, s_i^{attractor})$ starting from any reachable state $s \in R_i \cap G_{\mathcal{S}}$ towards the corresponding attractor state $s_i^{attractor}$ is complete.*

Proof. We can prove this lemma by contradiction. Firstly the greedy search in the query phase will always expand states which are *reachable* because the greedy successor of

Algorithm 3 Query

```
1:  procedure FindGreedyPath($s_1, s_2$)
2:      $s \leftarrow s_1$
3:      while $s \neq s_2$ do
4:          $s'_g \leftarrow \arg\min_{s' \in \text{Succ}(s)} h(s, s_2)$        ▷ greedy succesor
5:          $s \leftarrow s'_g$
6:          $\pi \cup \{s\}$
7:      return $\pi$
8:  procedure Query($s_{\text{goal}}$)
9:      for each $R_i \in \mathcal{R}$ do
10:         $(s_i^{\text{attractor}}, r_i) \leftarrow R_i$;
11:         if $h(s_{\text{goal}}, s_i^{\text{attractor}}) < r_i$ then
12:             $\pi_g \leftarrow$ FindGreedyPath ($s_{\text{goal}}, s_i^{\text{attractor}}$)
13:             return $\pi_g \cup \pi_i$        ▷ $\pi_i \in \mathcal{L}$
```

every reachable state is also reachable (This is by the construction of Alg. 2). If we show that the greedy search will not encounter any cycles or in other words it will never reexpand a state, we prove that the search will eventually reach $s_i^{\text{attractor}}$ and is hence complete.

For the greedy search to get stuck in a cycle, the greedy successor of every state in the loop must also be a part of the loop, or in other words there cannot exist a state in the loop with its greedy successor exiting the loop. But that is not possible since at least the first state in this loop that got added to $S_{\text{reachable}}$ must have a greedy successor from outside the loop and must also belong to $S_{\text{reachable}}$ with a valid edge connecting to it (Alg. 2, lines 10 and 11). Hence the theorem is proved. □

Lemma 2. *For any subregion $R_i \in \mathcal{R}$, every state s with $h(s, s_i^{attractor}) < r_i$ is reachable.*

Proof. We will again prove this lemma by contradiction. Assume that there exists a state s with $h(s, s_i^{\text{attractor}}) < r_i$ and was still not marked as *reachable* in the reachability search. In that case along the path from s to $s_i^{\text{attractor}}$, traced by the greedy search there must exist at least one state s' s.t. $h(s', s_i^{\text{attractor}}) > r_i$. The reason is that as the reachability search expands states in the order of minimum heuristic value, s' will never get expanded and thus will block the search from covering s. But since we assume that the heuristic function is weekly monotonic, this hypothesis cannot be true and hence there cannot exist such a state s. □

Theorem 1. *For any goal state $s_{goal} \in R_i \cap G_{\mathcal{S}}$ (i.e $h(s_{goal}, s_i^{attractor}) < r_i$), a greedy search with respect to $h(s, s_i^{attractor})$ over $\mathcal{S}$ starting at s_{goal} will result in a collision-free path to $s_i^{attractor}$.*

Proof. The proof directly follows from the two lemmas. If for any goal state s_{goal}, $h(s_{\text{goal}}, s_i^{\text{attractor}}) < r_i$, then s_{goal} is reachable (Lemma 1) and the greedy search towards $s_i^{\text{attractor}}$ is complete (Lemma 2). Also as Alg. 2 ensures that the transitions to the greedy successors are always valid we can guarantee that the greedy search will always return a valid path from s_{goal} to $s_i^{\text{attractor}}$. □

	PRM (T)	PRM (2T)	PRM (4T)	Our method
Planning time [ms]	20.2 (28.2)	21.8 (34.1)	22.9 (31.9)	0.48 (1.01)
Success rate [%]	92	97	100	100

Table 1: 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).

Lemma 3. *Alg. 2 returns the maximum radius r_i for a subregion R_i within which every state s with $h(s, s_i^{attractor}) < r$ is reachable.*

Proof. May be it is obvious and should just be added as a note and not a theorem. □

4.1 Time Complexity of Query Phase

Add branching factor.

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 the states along the path from s_{goal} to $s_i^{\text{attractor}}$ which requires $O(\mathcal{D})$ expansions where $\mathcal{D}$ is the depth of the deepest subregion. We can measure the depth of each subregion in Alg. 2 by keeping track of the depth of each expanded state from the root i.e. the $s_i^{\text{attractor}}$. Hence, overall the query phases takes $O(|\mathcal{R}| + \mathcal{D})$ operations. The maximal query time can also be empirically computed after the preprocessing phase.

Note that we can also bound the number of expansions required for the query phase by bounding the maximum depth of the subregions. We can do that by terminating Alg. 2 when the R_i reaches the maximum depth or if the existing termination condition (line 12) is satisfied.

5 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 1) 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.

6 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

[Berenson, Abbeel, and Goldberg 2012a] Berenson, D.; Abbeel, P.; and Goldberg, K. 2012a. A robot path planning framework that learns from experience. In *IEEE International Conference on Robotics and Automation (ICRA)*, 3671–3678.

[Berenson, Abbeel, and Goldberg 2012b] Berenson, D.; Abbeel, P.; and Goldberg, K. 2012b. A robot path planning framework that learns from experience. In *IEEE International Conference on Robotics and Automation (ICRA)*, 3671–3678.

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

[Conner, Choset, and Rizzi 2006] Conner, D. C.; Choset, H.; and Rizzi, A. 2006. Integrated planning and control for convex-bodied nonholonomic systems using local feedback control policies. Technical Report CMU-RI-TR-06-34, Carnegie Mellon University, Pittsburgh, PA.

[Conner, Rizzi, and Choset 2003] Conner, D. C.; Rizzi, A.; and Choset, H. 2003. Composition of local potential functions for global robot control and navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 4, 3546–3551.

[Dobson and Bekris 2014] Dobson, A., and Bekris, K. E. 2014. Sparse roadmap spanners for asymptotically near-optimal motion planning. *IJRR* 33(1):18–47.

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

[Kavraki et al. 1996] Kavraki, L. E.; Švestka, P.; Latombe, J.-C.; and Overmars, M. H. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automation* 12(4):566–580.

[Kavraki, Kolountzakis, and Latombe 1998] Kavraki, L. E.; Kolountzakis, M. N.; and Latombe, J. 1998. Analysis of probabilistic roadmaps for path planning. *IEEE Trans. Robotics and Automation* 14(1):166–171.

[Koenig and Likhachev 2006] Koenig, S., and Likhachev, M. 2006. Real-time adaptive A*. In *International joint confer-*

ence on Autonomous agents and multiagent systems, 281–288. ACM.

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

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

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

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

[Murray et al. 2016] Murray, S.; Floyd-Jones, W.; Qi, Y.; Sorin, D. J.; and Konidaris, G. 2016. Robot Motion Planning on a Chip. In *Robotics: Science and Systems (RSS)*.

[Paden, Nager, and Frazzoli 2017] Paden, B.; Nager, Y.; and Frazzoli, E. 2017. Landmark guided probabilistic roadmap queries. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4828–4834.

[Phillips et al. 2012] Phillips, M.; Cohen, B. J.; Chitta, S.; and Likhachev, M. 2012. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems (RSS)*.

[Phillips et al. 2013] Phillips, M.; Dornbush, A.; Chitta, S.; and Likhachev, M. 2013. Anytime incremental planning with e-graphs. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2444–2451.

[Salzman et al. 2014] Salzman, O.; Shaharabani, D.; Agarwal, P. K.; and Halperin, D. 2014. Sparsification of motion-planning roadmaps by edge contraction. *IJRR* 33(14):1711–1725.

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

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

[Yang et al. 2018] Yang, Y.; Merkt, W.; Ivan, V.; Li, Z.; and Vijayakumar, S. 2018. HDRM: A Resolution Complete Dynamic Roadmap for Real-Time Motion Planning in Complex Scenes. *IEEE Robotics and Automation Letters* 3(1):551–558.