Learning to Avoid Local Minima in Planning for Static Environments

Shivam Vats
Indian Institute of Technology, Kharagpur
shivamvats@iitkgp.ac.in

Venkatraman Narayanan and **Maxim Likhachev**
The Robotics Institute, Carnegie Mellon University
{venkatraman, maxim}@cs.cmu.edu

Abstract

In many robot motion planning problems such as manipulation planning for a personal robot in a kitchen or an industrial manipulator in a warehouse, all motion planning queries are in an environment that is largely static. Consequently, one should be able to improve the performance of a planning algorithm by *training* on this static environment ahead of operation time. In this work, we propose a method to improve the performance of heuristic search-based motion planners in such environments. The first, learning, phase of our proposed method analyzes search performance on multiple planning episodes to infer local minima zones, i.e, regions where the existing heuristic(s) are weakly correlated with the true cost-to-go. Then, in the planning phase of the method, the learnt local minima are used to modify the original search graph in a way that improves search performance. We prove that our method preserves guarantees on completeness and bounded suboptimality with respect to the original search graph. Experimentally, we observe significant improvements in success rate and planning time for challenging 11 degree-of-freedom mobile manipulation problems.

1 Introduction

A large number of robot motion planning problems occur in unchanging environments. For example, planning for a manipulator in an automated warehouse always involves generating motions that move the arm from one shelf bin to another, or to a drop-off region. Similarly, planning for a personal assistant robot involves generating paths with respect to a 3D map of the home that is mostly constant. In such scenarios, one naturally desires to exploit the static nature of the environment to improve motion planning efficiency. While memorizing all possible planning queries ahead of time would be the ideal solution, it is clearly intractable given the size of the configuration space—for example, even for a 3D (x, y, yaw) planning problem in a 100×100 grid with 10 discrete values for yaw, one would need to store 10^{10} possible plans. The challenge of learning a compact generalization to improve planning efficiency in such environments motivates this work.

We focus on heuristic-search based motion planning on lattice graphs representing the configuration space. Our key observation is that the efficiency of heuristic search methods is mainly affected by large "local minima" regions (regions where the heuristic estimate is weakly correlated with the true cost-to-go) in the environment, and that a post hoc analysis of a heuristic search episode will reveal these regions. Based on this insight, we propose a two-phase algorithm: i) the first, learning phase, collects search statistics (specifically, time at which every state was expanded during the search) for several planning episodes to determine local minima regions, ii) the second, planning phase, uses the learnt local minima to construct a query-specific graph for improving planning efficiency. We prove that our approach preserves theoretical guarantees on completeness and solution quality with respect to the original lattice graph. Experimentally, we show that our method significantly improves success rate and planning time for 11 DoF mobile manipulation planning problems.

2 Related Work

The idea of improving best-first search planning efficiency by avoiding local minima has been explored in several works. There are two approaches that are commonly adopted. In the first, search algorithms are modified to detect local minima or plateau regions and then use random exploration (Valenzano and Xie 2016) or a sub-goal search (Likhachev and Stentz 2008; Chakrabarti, Ghose, and Desarkar 1986) to escape the local minima. The second approach focuses on learning a new heuristic based on prior planning episodes to effectively circumvent local minima in the future (Phillips et al. 2012; Xu, Fern, and Yoon 2007). In contrast to both these approaches, our proposed algorithm attempts to modify the search graph itself so that existing heuristics can perform better on the new graph. The most relevant prior work to our algorithm is Marvin (Coles and Smith 2007), which learns new macro-actions for the graph based on prior planning episodes. The key difference between Marvin and our algorithm is that the former modifies the graph by introducing edges that are always a composition of existing edges, whereas we do not have that restriction. This efficiency improvement is possible because of the continuous state space specific to robot motion planning.

3 Algorithm

We assume that the motion planning problem is represented as a graph search problem on an implicit graph defined by a successor function SUCC. The vertices of the graph are states (denoted by s) in the robot configuration space and edges between two states are *motion primitives*, which are kinodynamically feasible motions between those states. In addition to using a fixed set of precomputed motion primitives to construct the graph, we will also allow for the use of *adaptive motion primitives* (Cohen et al. 2011) for generating new motions/edges on the fly during graph construction. In kinematic planning for example, an adaptive motion primitive can be generated by simply interpolating between the source and target states and verifying that the motion is collision-free and satisfies the robot's kinematic constraints.

Overview. Our algorithm has two phases: a learning phase and a planning phase. In the first phase, we run a heuristic search planner on a number of training data points (randomly selected start-goal pairs in the environment) and analyze the results to learn a compact generalization named "activation regions" to capture the local minima in the environment. Next, the planning phase uses the learnt activation regions to modify the graph on-the-fly to efficiently find a solution for new test cases. One could also close the loop and treat the new test cases as additional training data points, to continuously improve performance.

Learning Phase. Assume that we run a heuristic search planner such as A* or Weighted A* (Pohl 1970) on the lattice graph. Admissible heuristics are often imperfect and can mislead the search, causing it to expand states that will eventually never be part of the returned solution. The regions where unnecessary states are expanded are the local minima zones. Ultimately, we want to identify these regions during the learning phase to help improve performance later. Our intuition for the learning phase is that by determining the states on the solution path that took "longer" to discover, we can identify the local minima regions where the search spent time.

Formally, let $\{s_1, s_2, \ldots, s_N\}$ be the ordered set of states expanded by the search for a particular start-goal pair, with $N' \leq N$ states on the solution path. For each state s_i expanded by the search, we will also record the timestamp t_i at which it was expanded. Now, consider two consecutive states on the solution path, s_i and s_j, with timestamps t_i and t_j respectively. If the difference between t_j and t_i were "small', then it implies that the heuristic gradient was informative at state s_i. Conversely, if the time difference were large, then the search must have spent a significant amount of time expanding states in a local minimum. It follows then, that the time difference $\Delta t_{ij} = t_j - t_i$ is a measure of the depth of local minimum in that region of the state space.

Figure 1 shows an example plot of the first difference of the expansion times $\Delta t_{i(i-1)}$ (abbreviated Δt_i) against the N' states in the solution path. The peaks in the plot correspond to the states that took longest to discover, and are also "exits" to local minima in the corresponding region of the state space. Algorithm 1 shows how to discover these states.

Algorithm 1 Learning Phase

```
1: procedure FINDLOCALMINIMA(m)
2:     P ← random start-goal pairs                ▷ Training Set.
3:     for p_i ∈ P do
4:         success = RUNPLANNER(p_i)
5:         if success then
6:             T = timestamps of all states on path
7:             for each state s_i on path do
8:                 calculate Δt[i] = T[i] − T[i − 1]
9:     return m highest local maxima from the Δt curve
```

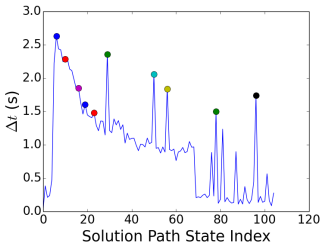

Figure 1: A plot of Δt, the first difference of the expansion times, versus the states on the solution path. The colored dots indicate the top 10 local maxima that correspond to the learnt activation centers.

The next task is to learn a generalization from the exit states so that they can be used for similar planning queries in the future. To do so, we will approximate the local minima as hyperspheres in the configuration space called *activation regions*. Since the search would benefit most from knowing how to escape each local minima, we will define an activation region a_i as centered at an exit state s_i, and having a radius r_i. Algorithm 1 describes the procedure to compute the activation region centers s_i, which in turn correspond to the peaks of the curve in Fig. 1.

Next, we obtain the activation region radii r_i by essentially computing the half-widths of the peaks in Fig. 1. Starting at an exit state, s_i we iterate backwards and look at the second difference of the expansion times (i.e, $\Delta t_j - \Delta t_{j-1}$). The state, s_k at which this changes from positive to negative does not belong to the current local minimum, and we mark it as its beginning. Consequently, we define $r_i = \alpha \cdot \text{DIST}(s_k, s_i)$, where DIST is a domain dependent distance function, and α is a variable inflation factor.

Planning Phase. Consider a planning problem with start-goal pair $p = (s_{start}, s_{goal})$. Having learnt activation regions for the whole environment and for a variety of training data points, we would first like to select those activation regions that cover the local minima likely to be encountered while trying to solve p. To do so, we choose all activation regions that were encountered by the planner while solving problems that were "similar" to p during the learning phase. We measure distance between two planning problems by summing the domain dependent distances between their respective start and goal states: $\text{DIST}(s_{start}, s'_{start}) + \text{DIST}(s_{goal}, s'_{goal})$. While s_{start} does not influence the cost-to-go, using similarity between both start and goal restricts the number of relevant activation regions.

If we choose to select n most similar training problems, we get a set of $m \cdot n$ activation regions, where m is the number of activation regions learnt per training episode. Next,

Algorithm 2 Successor Function for Planning Phase

```
1: procedure AdaptiveSuccs(s)
2:     succs ← Succs(s)                          ▷ original successors
3:     for a ∈ activation regions do
4:         if s ∈ a then
5:             c ← a.center                      ▷ "exit" state
6:             motion = AdaptiveMotion(s,c)
7:             if motion is feasible then
8:                 insert c in succs
9:     return succs
```

we describe our approach to using these activation regions during planning. We run the same search algorithm as used during the training phase, albeit with a modified successor function given in Alg. 2. Every time we expand a state that is within an activation region, we augment the regular set of successors with the center of the activation region (i.e, the local minima's exit state). The adaptive motion between the expanded state and the exit state could be as simple as straight-line interpolation, or an arbitrarily complicated one. The idea behind this adaptive successor generation is to preemptively prohibit the search from expanding states in the local minima. We term the implicit graph resulting from this modified successor function as the *Adaptive Motion Graph*.

Finally, we note that the choice of n and m for a problem is largely a function of the environment's complexity and the heuristic's quality. A very small m could lead to poor learning, while a large value could result in significant overhead during the planning phase.

Theorem. *The solution returned by a search algorithm on the adaptive motion graph is optimal (bounded suboptimal) with respect to the original graph, if the search algorithm is optimal (bounded suboptimal) on the original graph.*

Proof. (Sketch.) This theorem follow from the observation that the original graph is a subgraph of the adaptive motion graph. Since the minimum of a set is always upper bounded by the minimum of any of its subsets, the property holds. □

Following a similar reasoning, if the search algorithm is complete on the original graph, then it remains complete with respect to the adaptive motion graph.

4 Experimental Results

We evaluate the benefits of our learning approach on a challenging 11 DoF mobile manipulation planning problem for the PR2 robot, the same domain used in (Narayanan, Aine, and Likhachev 2015). We also use the Focal-MHA* search algorithm from the same paper, as it was shown to perform well on this domain. Results are presented for running Focal-MHA* on the adaptive motion graph as well as the original graph. In addition, we also present comparisons with a popular sampling-based motion planning algorithm, RRT-Connect (Kuffner and LaValle 2000).

Domain. The PR2 robot is a mobile manipulation platform with two arms, a telescoping spine, and a mobile base. We restrict ourselves to planning only for a single arm, the spine, and mobile base. Each 11 DoF state in the graph we plan on is comprised of a 6 DoF object pose for the end-effector $(x, y, z, roll, pitch, yaw)$, one redundant free angle for the right arm, a prismatic degree of freedom for the spine, and (x, y, yaw) for the robot base. The start state for the planner s_{start} is a fully specified 11 DoF state, whereas the goal state s_{goal} is an underspecified 6 DoF end-effector configuration. Consequently, any state that results in the end-effector of the robot reaching the object's pose meets the goal condition. In addition to the default motion primitives described for this domain, we define two types of adaptive motions for the AdaptiveMotion method in Alg. 2:

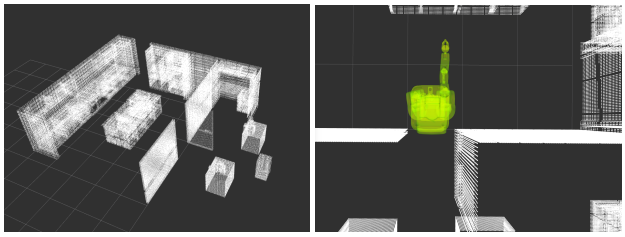

Figure 2: *Left:* Planning environment. *Right:* A learnt activation center near the doorway—a deep local minimum for the heuristics.

- **Full-body Snap Motion Primitive:** This adds the activation center (a full 11 DoF state) to the open list by interpolating a straight line from the expanded state to the activation center. We use a low α (2) as arm motions over long distances are likely to collide with the environment.
- **Base Snap Motion Primitive:** This adds a state with the base configuration of the activation center but arm and torso configuration of the source state. Here, we can afford to use a higher α (15) since there are no arm motions, and we can take advantage of long-range base motions.

Note that by using motion-primitive specific activation region radii, we can preemptively eliminate adaptive motions that are unlikely to be valid.

Implementation Details. Focal-MHA* is a multi-heuristic search algorithm which uses one consistent heuristic and possibly several inadmissible heuristics. It is guaranteed to provide a solution which is at most w suboptimal, and does not expand a graph state more than twice. We can used this property to optimize successor generation in Alg. 2 in the following manner: we maintain two separate sets of activation centers—one for the admissible anchor heuristic and another for the inadmissible heuristics. Once an activation center is expanded by an admissible or inadmissible heuristic, we delete it from the corresponding set. This reduces the branching factor of the search when possible. In all our experiments, we used 60 activation regions ($n = 4$ and $m = 15$). For the domain dependent distance measure Dist between two states, we use the L_∞ distance between their corresponding 3 DoF (x, y, θ) robot base configurations.

For Focal-MHA*, we used the same set of 20 heuristics used in (Narayanan, Aine, and Likhachev 2015). Addition-

	$w = 20$			$w = 50$		
	Adaptive	Original	RRT-C	Adaptive	Original	RRT-C
Success Rate (%)	75	69	38	74	69	38
Speedup	1	1.85	0.81	1	1.6	0.59
Planning Time (s)	54	77	44	55	77	44

Table 1: Focal-MHA* with 10 heuristics was used to search on the original and the adaptive motion graphs.

	$w = 20$		$w = 50$	
	Adaptive	Original	Adaptive	Original
Success Rate (%)	67	46	69	47
Speedup	1	1.98	1	2.1
Planning Time (s)	71	100	74	98

Table 2: Focal-MHA* with 20 heuristics was used to search on the original graph as well as the adaptive motion graph.

ally, to show the generality of our learning approach, we repeat all experiments with a) a set of only 10 heuristics (by dropping all base rotation heuristics) and b) using suboptimality factors of 20 and 50 for Focal-MHA*.

For our experiments, we used a typical house environment with a kitchen and a room connected via a doorway. We also generated 3 variations of this environment by adding two tables in the room at random positions along with a different number of objects on the tabletop (at random locations). The learning phase is repeated for each variant, and results on testing instances are averages across all environments. The heuristics used are variants of two main types: base heuristics that guide the robot's (x, y) location to the goal region, and end-effector heuristics that guide the (x, y, z) of the end-effector to the goal. Looking at the environment in Fig. 2, one might guess that regions of deep local minima are likely to occur at the door, around tables and around clutter on top of these tables. The activation centers that were learnt during our experiments confirm our intuition. Fig. 2 shows such an activation center at the door. Note how this state has the "correct" exit base orientation and arm configuration to get through the door, despite having no heuristic that provides this information. Now whenever the planner receives a query in which the robot needs to get through this door, an adaptive motion primitive will be added as an edge to this state.

Results. We generated 120 random trials for each environment. Out of these, 60 were used in the learning phase and 60 for testing performance. Each trial was created by choosing a random pose on one of the two tables for the end-effector to reach and a randomly generated starting configuration for the robot. A trial is successful if the planner finds a w-optimal solution within the time limit. Time limit was set to 200 seconds when planning with 10 heuristics and to 300 seconds when using 20 heuristics. All experiments were run on an Intel i7-6700HQ CPU (2.60 GHz) with 16GB RAM.

Results of our experiments are summarized in Table 1 and Table 2. The algorithms are compared against the measures of success rate and mean planning time. Further, we calculate a measure of speedup by taking the geometric mean of the ratios of planning times. These statistics are computed

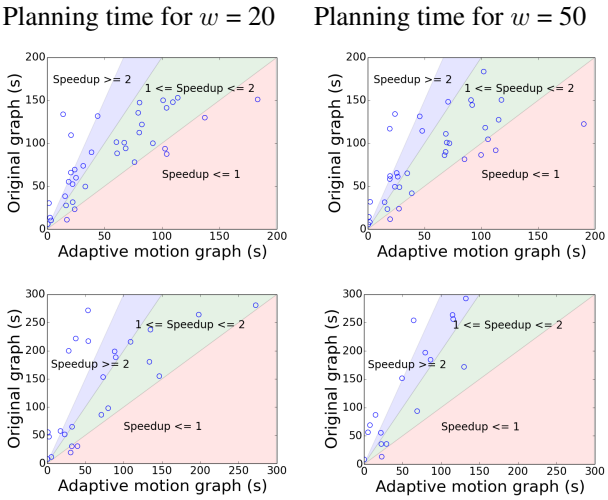

Figure 3: The scatter plots compare the time taken by Focal-MHA* on an adaptive motion graph with that taken on the original graph on every instance. The different shaded regions represent distinct speedup intervals. The top two plots are with 10 heuristics and the bottom two are using 20 heuristics.

on only those trials in which adaptive motion graph and the algorithm being compared to, both succeeded. The main takeaway is that our method shows marked improvement in planning time and success rate over using the original graph, across all values of w, the number of heuristics used, and different environments.

From Table 1, we see that the RRT-Connect has a better planning time than our method. However, its success rate is drastically smaller, primarily owing to the narrow doorway in the problem. The problems it does solve are the easier ones, where our adaptive motion graph incurs unnecessary overhead. However, as can be seen in the scatter plots in Fig. 3, there are very few trials in which the adaptive motion graph does worse than the original graph, implying that its overhead is not as severe.

5 Conclusions

We presented an algorithm for improving planning efficiency in static environments by learning local minima regions and using them to modify the search graph. In addition to being conceptually simple, our method was shown to provide significant improvements in success rate and planning time over a non-learning approach, for a challenging 11 DoF mobile manipulation planning domain. For future work, we are interested in generalizing our approach to domains where the environments are not identical but do share some common "features". Consequently, we would require automatic feature learning for transferring the local minima knowledge across different environments. Finally, we would like to automate learning of the activation region parameters (e.g. α) to dynamically adapt their shape and reduce the overhead induced by additional graph edges.

Acknowledgments

This work was supported by NSF grant IIS-1409549 and by ARL under the Robotics CTA program grant W911NF-10-2-0016.

References

Chakrabarti, P.; Ghose, S.; and Desarkar, S. 1986. Heuristic search through islands. *Artificial Intelligence* 29(3):339–347.

Cohen, B. J.; Subramania, G.; Chitta, S.; and Likhachev, M. 2011. Planning for manipulation with adaptive motion primitives. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 5478–5485. IEEE.

Coles, A., and Smith, A. 2007. Marvin: A heuristic search planner with online macro-action learning. *J. Artif. Intell. Res.(JAIR)* 28:119–156.

Kuffner, J. J., and LaValle, S. M. 2000. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, 995–1001. IEEE.

Likhachev, M., and Stentz, A. 2008. R*Search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Narayanan, V.; Aine, S.; and Likhachev, M. 2015. Improved multi-heuristic a* for searching with uncalibrated heuristics. In *Eighth Annual Symposium on Combinatorial Search*.

Phillips, M.; Cohen, B. J.; Chitta, S.; and Likhachev, M. 2012. E-graphs: Bootstrapping planning with experience graphs. In *Robotics: Science and Systems*, volume 5.

Pohl, I. 1970. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* 1(3-4):193–204.

Valenzano, R. A., and Xie, F. 2016. On the completeness of best-first search variants that use random exploration. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Xu, Y.; Fern, A.; and Yoon, S. W. 2007. Discriminative learning of beam-search heuristics for planning. In *IJCAI*, 2041–2046.