2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

PLRC*: A piecewise linear regression complex for approximating optimal robot motion

Luyang Zhao Josiah Putman Weifu Wang Devin Balkcom

***Abstract*— Discrete graphs are commonly used to approximately represent configuration spaces used in robot motion planning. This paper explores a representation in which the costs of crossing local regions of the configuration space are represented using piecewise linear regression (PLR). We explore a few simple motion planning problems, and show that for these problems, the memory required to store the representation compares favorably to that required for standard discrete vertex-and-edge models, while preserving the quality of paths returned from searches.**

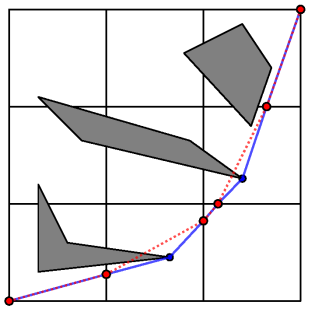
I. Introduction

The development of the PRM* and RRT* and related algorithms has shown that in principle, sampling-based planners can return trajectories that approach the minimum cost for some metric as the number of samples approaches infinite. But how much memory and computation time are really needed to adequately approximate optimal motion? Even in an empty space, we expect a PRM* graph to require dense sampling, since samples must be near the optimal path; the present work attempts to exploit smoothness of the metric to build approximations that do not need such dense sampling.

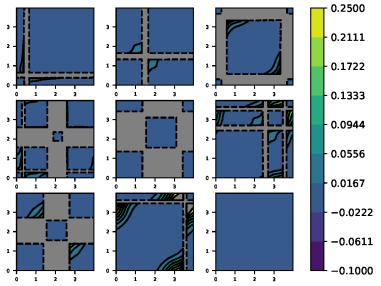
This paper presents a data structure, the regression complex (RC), for summarizing approximately optimal motion in robot configuration spaces. While the approach does not escape the curse of dimensionality, it does take advantage of smoothness in the configuration-space metric to build cells that summarize information that might otherwise require many samples. Regression techniques are used to express the distance between boundary points of each cell, smoothly approximating the cost function that would otherwise be implicitly represented using a large number of edges between samples within each cell. Because cells are large, the true cost of a path can be estimated quickly in a query phase.

Memory-efficient data structures for motion planning are important. As cloud computing and faster processors make computation more available, memory is often the bottleneck in motion planning. Consider any robot supported by cloud infrastructure, which needs the brunt of its motion planning computed remotely. Large data structures like PRMs can be constructed using offline preprocessing, but to make the information available to robots locally requires storage or transmission across a network. The present work provides a light-weight data structure that can be constructed remotely, transmitted once, and used by the robot for multiple planning queries. The approach uses a divide-and-conquer strategy, allowing separate regions of the search space to be processed sequentially (to reduce memory) or in parallel (for speed). The parallelizability of the construction procedure also has promising applications using distributed computing.

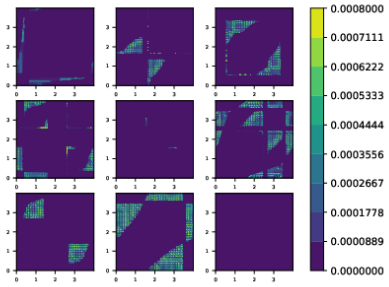
A. A simple example



(a) 2D environment



(b) Cell distance functions



(c) Error between prediction and ground truth

(1) Complex with distance calculated from visibility graph

Consider the trivial example of a planar space for a point robot with polygonal obstacles in Figure 1a. We have broken the space into a 3×3 grid of square cells. We parameterize the boundary S of each cell with a real number in the range $[0, 4)$. Let the cost function between points on the boundary be $f_i : S \times S \mapsto \mathbb{R}$, where i is the index of the cell. Figure 1b shows each of the boundary cost functions for the nine cells.

These functions are not defined everywhere, since some paths may not exist; these regions are shown in gray. The red dots in Figure 1a shows points along a path extracted from the distance functions using techniques from the paper, and the blue path shows a refinement of the path that avoids obstacles within the cells.

Figure 1b is *not* a regression complex, but was generated by sampling the visibility graph [14], which can be used to find optimal paths in planar environments with polygonal obstacles and a local Euclidean metric. The paper will describe how such distance functions may be approximated using regression techniques, how paths may be extracted using the data structure, and evaluates the usefulness of the data structure. Figure 1c shows the error between approximated distances and ground truth, where the ground truth is computed using a visibility graph. The RMSE is 0.0001658 in this case.

B. Weaknesses and limitations

This paper makes the same strong assumptions as PRM*: that the configuration space is fully known, and that there is a local planner that can connect pairs of nearby configurations optimally without too much computational expense.

The present work is preliminary, and provides at best an exploration of an alternative to popular sample-and-edge discretizations of configuration spaces. How should these regression complices be constructed efficiently? How should they best be searched for a path? Once a path is found, how can it be refined to avoid obstacles within each cell? We don't know. For each of these phases of *construction*, *query*, and *refinement*, we have explored only a first approach. For example, in the construction phase, we use a PRM* to measure the underlying cell-crossing metric that the regressions approximate, though we are certain that a PRM* is costly and unwieldy for the purpose. We focus on a piecewise linear regression technique for transparency, but can imagine that other techniques (such as neural networks) might be as or more effective. Similarly, in the query phase, we discretize cell-boundaries and search, though the existence of a smooth approximation of the metric strongly suggests an optimization approach like that used by Field D* [10].

Experimental work, though promising, is limited so far to examples in a few dimensions, due in part to the computational cost of sampling the value functions exhaustively.

II. Related work

Work in the present paper attempts to pre-compute and compactly represent cost and connectivity information about robot state or configuration spaces for motion planning, dividing search into *learning* and *query* phases, just as sampling-based [12] approaches do. The PRM* [11] algorithm and its relatives have shown that as samples are placed densely enough, with enough connecting edges and an optimal local planner, sampling-based planning methods can approximate optimal motion with high probability. However, this approximation comes with a memory cost – in an example in [11], the PRM* algorithm continues to meaningfully improve upon the path cost while 50000 samples are placed, in a planar environment for a point robot with no obstacles. We can imagine why: a sampling-based planner must place samples quite finely within a tube surrounding a minimum-cost trajectory to approximate that trajectory, and PRM* must effectively do so for every potential pair of start and goal configurations.

Work on graph spanners [15, 13, 22] has shown that many of the edges in such graphs can be deleted without too much harm to the path quality. Our own prior work [1] finds formal upper bounds on the complexity of approximating optimal trajectories through cell decomposition, but the memory costs, while finite and computable for any given space, are high.

Approaches that remove or avoid placing vertices, such as the Visibility PRM [20] and certificate methods [4, 8], must sacrifice optimality. Similarly, trees or graphs of controllers (e.g. LQR trees [21]) may cover large regions of space nicely, but are focused on robust control rather than optimal path cost. In [6], a metric for swept volume is learned, and serves as an effective and admissible heuristic for planning.

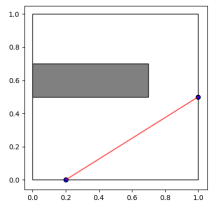
Smooth approximations of value functions are quite common in reinforcement learning, but have also made some appearance in robot motion planning. For example, work by Rayner, Bowling, and Sturtevant [18], remaps a motion problem with obstacles in such a way that Euclidean distance in the warped map serves as a heuristic for the original problem. Recent work by Faust *et al.* [9] explores a combined sampling and reinforcement-learning approach to long-range planning, addressing some of the same computational issues as the present paper, but with less focus on optimal motion. Network embeddings also attempt to find a function that expresses distance between vertices in a network; [7] provides a recent survey.

Neural networks have been used to learn value functions derived from optimal control; one of the first examples is [16]. The present work uses a simple piecewise linear regression in the interest of transparency, and also attempts to represent an all-pairs problem, rather than a one-shot planner to a single goal configuration. Distance metric approximation for RRTs using supervised learning [3] has been used for tasks such as pendulum swing-up problems, but is likewise focused on one-shot planning.

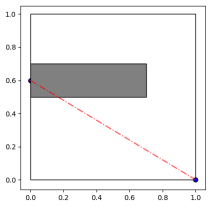
Our previous work on Locally Linear Distance Maps (LLDM) [17] explored using piecewise linear regressions of the distance-to-goal as a heuristic for existing motion planning algorithms. Instead of using the linear regressions as a heuristic, the regression complex uses the same regression method to approximate boundary-to-boundary costs in each cell of the complex.

III. The regression complex

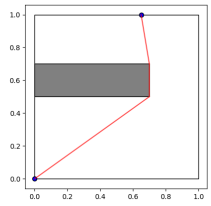
In this section, we describe an approach to constructing and querying the regression complex (RC) data structure using a piecewise linear regression to approximate cell-crossing costs.



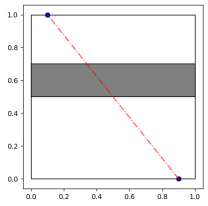
(a) Metric given by local planner.



(b) One point in collision.



(c) Regression used to approximate metric.



(d) Boundary points in separate components.

(2) Four different relationships among boundary points

We will need some notation. Let $Q_f \subseteq Q$ denote the free regions of Q. Let $\Delta : Q_f^2 \mapsto \{0,1\}$ be a local planner that determines if two states $q_1, q_2 \in Q_f$ are connectable. If the states are connectable, we assume that the local planner is also capable of providing the cost of an optimal path between q_1 and q_2. Let $d : Q^2 \mapsto \mathbb{R}_{\geq 0}$ denote the distance metric between pairs of states.

A. PLRC*: Construction phase

The construction phase decomposes Q into a set of cells $\mathcal{C}$, where each cell $C \in \mathcal{C}$ is an n-dimensional hypercube. For each cell, the basic approach is to sample distances between pairs of points along the boundary of the cell, and then to use this data to build a regression.

Let $B(C)$ represent the boundary of a cell C. We associate with each cell C a regression $R_C : B(C) \times B(C) \mapsto \mathbb{R}_{\geq 0}$ that maps pairs of points on its boundary to the distance between them. A classifier $K_C : B(C) \times B(C) \mapsto \{0,1\}$ is also constructed to represent the connectivity of pairs of boundary points.

We enumerate four cases that the relationship between a pair of points on the boundaries of a cell may fall into, shown in Figure 2; we handle each of these cases differently in the representation of the regression complex data structure.

Two points on the boundary of a cell are connectable by a path that remains within a cell (2a, 2c), or they are not (2b, 2d). Although one could imagine the path cost between unconnectable points as infinite, the resulting discontinuities in the value function are problematic for a smooth regression representation.

We use two approaches to mask out unconnected configurations from the regression. First, we use a collision detector to find if either boundary point is itself in collision with an obstacle. If so, we discard the sample. Second, if the boundary points themselves are collision-free, but the local planner fails to find a path connecting the points *without leaving the cell*, we record this fact using a classifier, the implementation of which we discuss below.

Restriction of the local planner to find only paths completely constrained to the current cell is an important design decision for the approach, and means that the regression does not encode the true value function between each pair of boundary points. Instead, some search over adjacent cells may be needed to connect two points on the boundary over a cell, and a cell may need to be crossed several times by a path found in the query phase. Allowing this to happen separates the problem of determining cell size from the geometry of the obstacles, allowing much larger cells than those used in typically cell-approximation methods for motion planning (*e.g.* [1, 2]).

We also introduce an optimization to exploit the fact that for paths that do not contact obstacles (Figure 2a), the local planner already provides an accurate local metric. In some cases, the local planner provides a useful estimate even if a path gently grazes an obstacle. Therefore, rather than representing the cell-crossing distance function directly, we represent the difference between the cell-crossing distance function and the metric provided by the local planner. Thus, if the local planner is accurate over some region without obstacles, the regression needs only to store the value 0 over that region.

The training data used for the cost regression and the connectivity classifier might be found in various ways. For simplicity, and for simplicity of comparison with standard PRM* we make use of a local PRM* inside the cell.

B. Piecewise linear regression

The Piecewise Linear Regression (PLR) used to approximate cell-crossing distance functions is discussed in greater detail in a technical report [17], along with proofs of convergence and use cases. The technique is straightforward. We would like to approximate a function from $R^n \mapsto R$. The PLR subdivides the space into cells using a Binary Space Partition (BSP), sampling the function as it goes, and using the samples to compute a linear approximation in each cell of the BSP. The error may then be estimated using further samples; if needed, the BSP is refined further. We used the same structure with a threshold value of 0.5 as the cell's connectivity classifier K_C.

All experiments presented in this paper use the piecewise linear approach to regression and classification, since this approach is quite transparent, simple to implement, requires little tuning, and provides some guarantees that at least the samples selected during training will be fit well, giving us some confidence in the overall approach.

However, the regression complex framework is modular and any general-purpose regression technique can be applied; we are hopeful that for higher dimensions, other regression techniques will prove effective. We ran preliminary tests using both feed-forward deep neural networks and XGBoost [5]; results were useable but somewhat less robust in terms of quality of fit for similar memory costs, perhaps due

to our lack of insight into how to best design the architecture and tune the parameters.

C. PLRC*: Query Phase

Given a start configuration $s \in Q_f$ and a goal configuration $t \in Q_f$, a query is performed using a simple graph search with discretization along the boundaries, as described below:

1) Boundary graph construction: We construct a weighted graph $G_b = (V_b, E_b, w_b)$, whose vertices V_b are samples in Q_f along the boundary of every cell $C \in \mathcal{C}$. The edges E_b are pairs of points that are in the same cell C and are classified as connectable by K_C, i.e. $E_b = \{(q_1, q_2)$ s.t. $q_1, q_2 \in C, K_C(q_1, q_2) = 1$ for some $C\}$. The weight map w_b for each edge is calculated using the regressor, namely $w_b(q_1, q_2) = R_C(q_1, q_2)$. We then run A* search over G_b to return the path P_0, the unrefined path whose waypoints represent the points that the final path P needs to go through.

2) Path refinement: One can imagine many ways of reconstructing P from P_0. A PRM* could be constructed within each cell that P_0 passes through, and the path between each waypoint could be calculated using A* search. However, this approach requires constructing an incredibly dense PRM*, which is costly in terms of memory and defeats the purpose of using this algorithm. Alternatively, an optimizer could be used to place intermediate points between each waypoint such that the path between the added points is collision-free and the cost is minimized. Our approach uses the optimizer, and incrementally starts by attempting the connection with a single intermediate point. If the connection fails, an additional point is added and the procedure continues until the points are connected. This is guaranteed to terminate because the points on the graph were classified as connectable by their cell's classifier K_C.

IV. Experiments

This section presents the results of several computational experiments in which PLRC data structures are constructed and queried. For comparison, we also construct and query PRM* graphs directly.

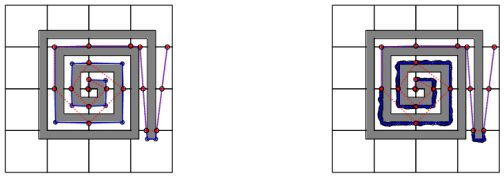
To measure the memory cost of each approach, we compute the number of floating points needed to store each data structure; this is a rough but perhaps useful estimate. For the PLRC, the memory cost includes the cost of both the regressor and the classifier. The cost of each is the sum of the number of numbers needed to store the parameters for each linear regression, the bounds of each cell in the BSP, and the number of cells needed, based on the depth of the BSP. The memory cost of PRM* is just the memory cost of storing the vertices and graph, which is $N * D + N * k * 3/2$, where N is the number of sample points used in the construction phase, and k equal to $\log(2, N)$.

A. Planar point robot among polygonal obstacles

We performed our first experiments for a planar point robot with $q = (x, y) \in \mathbb{R}^2$. The robot moves amongst polygonal obstacles from a start configuration to a goal configuration.

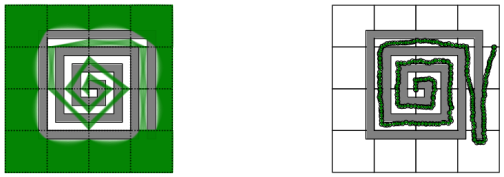
We tested several different environments, and present results (Table I) for two of the more interesting, which we name world-doors (Figure 3) and world-maze (Figure 4). We used different numbers of cells, ranging from 4 to 32 (PLRC*-4 to PLRC*-32).

From Table I, we can see that the memory required to store the PLRC data structure is about 20 times less than that required for the PRM* for these examples, and query of the PLRC returns a slightly more accurate approximation of the path cost. (The optimal path distance of maze-doors is 5.859918264 and of maze-world is 3.911704702 calculated from visibility graph[14].)



(a) PLRC*-16 with optimizer for reconstruction

(b) PLRC*-16 with PRM* for reconstruction



(c) Connection relationship between boundary points

(d) Path from PRM*

(3) World-maze: PRM* paths, and PLRC*-16 with different refinement techniques

Surprisingly, when the number of cells increases, PLRC* seems to require less memory. When we divide the regions into more small cells, the number of sample points become smaller and the distances of those points has lower variance; the piecewise linear regression subdivides these simpler smaller cells less.

The query phase of PLRC returns only the path cost and some points where the approximation of the optimal path intersects with cell boundaries. To find a higher-resolution path that avoids obstacles within cells, some refinement step is required, as described above.

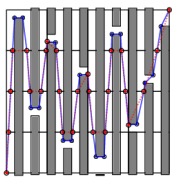
Figure 3a and Figure 4a show the path after refinement by optimization techniques and Figure 3b and Figure 4b show a path after refinement using a local PRM* built within each cell along the approximately optimal path. We note that the optimizer returns very good paths indeed, but the comparison to PRM* is not quite fair, since PRM* does not execute a refinement step. Figure 3d and Figure 4d shows the path computed by PRM* directly. Figure 3c and Figure 4c shows the connectivity of boundary points, as captured by the classifier.

Environment	Algorithm	Memory cost (FP)	Unrefined cost	Query time (sec)	Refinement method	Path cost	Refinement time
WORLD-DOORS	PRM*	215000	-	0.09644	-	6.14225	-
	PLRC*-4	9184	6.04741	0.01806	Optimizer	5.88847	161.16488
					PRM*	6.10913	41.43030
	PLRC*-16	4620	6.01461	0.00956	Optimizer	5.89986	14.75135
					PRM*	6.07112	18.71936
	PLRC*-32	3968	6.04388	0.00933	Optimizer	5.94593	12.99612
					PRM*	6.10998	12.47269
WORLD-MAZE	PRM*	215000	-	0.03401	-	4.09641	-
	PLRC*-4	21224	3.99711	0.00449	Optimizer	3.97747	1.65003
					PRM*	4.07569	19.37482
	PLRC*-16	5058	3.99963	0.00288	Optimizer	3.94168	2.61319
					PRM*	4.07431	7.29743
	PLRC*-32	4874	3.97164	0.00809	Optimizer	3.93096	15.92245
					PRM*	4.06221	9.44786

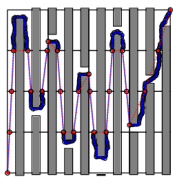
(I) Comparison between PRM* and PLRC* in different number of cells

Environment	Algorithm	Memory cost (FP)	Unrefined cost	Query time (sec)	Refinement method	Path cost	Refinement time
3D Arm in WORLD2	PRM*	210875	-	0.11388	-	5.44542	-
	PLRC*-8	21554	5.23471	0.04129	PRM*	5.40983	22.37812
RS Car in WORLD-DOORS2	PRM*	180000	-	0.083801	-	5.77792	-
	PLRC*-8	25122	5.48602	0.035062	PRM*	5.69049	33.10321

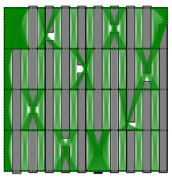
(II) Comparison between PRM* and PLRC* for 3D arm and RS car



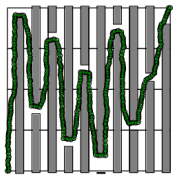
(a) PLRC*-16 with optimizer for reconstruction



(b) PLRC*-16 with PRM* for reconstruction

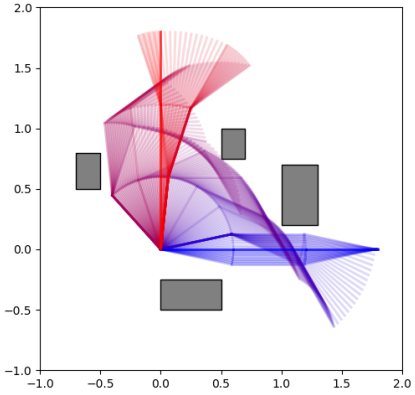


(c) Connection relationship between boundary points



(d) Path from PRM*

(4) World-doors: finding path with prm* and PLRC*-16 with different reconstruction techniques

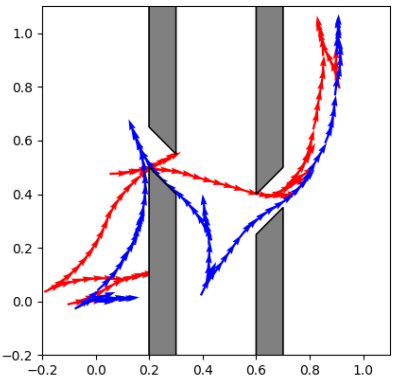


(5) 3-joint revolute arm path obtained by PLRC* from $(0,0,0)$ (blue) to $(\frac{\pi}{2},0,0)$ (red) in WORLD2. Total path cost 5.40983, compared to 5.44542 for PRM*.

B. n-joint revolute arm

Our next experiments were performed on an n-joint revolute arm whose configuration can be described by $q = (\theta_1, \theta_2, \ldots, \theta_n)$. For simplicity, we add a constraint $\theta_i \in [-\pi, \pi]\ \forall i \in [n]$.

We define the distance metric $d(q_1, q_2) = \max_{i\in[n]} |q_2[i] - q_1[i]|$, where $q[i]$ denotes the ith parameter of q. Note that if we assume that each joint has the same bound on its angular velocity, the above metric represents the time cost of going from configuration q_1 to q_2. The first row of Table II shows the comparison results of 3R arms from PRM* and PLRC*-8, where the memory cost of PRM* is 23 times of PLRC*-8 while the path accuracy of PLRC*-8 is much higher than PRM*. Figure 5 shows path obtained from PLRC*-8.



(6) Reeds-Shepp Car path obtained from PRM* (blue, path cost 5.77792) and PLRC* (red, path cost 5.69049)

C. Reeds-Shepp Car

We tested the algorithm on a Reeds-Shepp car [19] with obstacles which is showed in Figure 6. The second row of Table II shows the comparison results of Reeds-Shepp car from PRM* and PLRC*-8, where the memory cost of PRM* is 7 times of PLRC*-8 while the path accuracy of PLRC*-8 is higher than PRM*.

D. Comparison to spanner algorithms

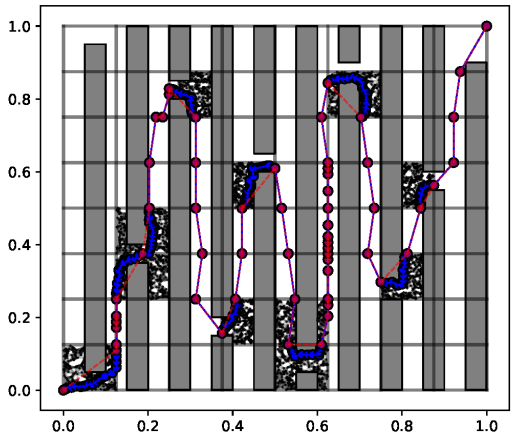
As a memory-efficient representation for motion planning, spanners can also save much space compared to traditional roadmaps, while returning *good quality* paths [15, 13, 22]. However, one of the major differences is that the spanners are sub-graphs of the roadmaps, though some approaches are online and never store the complete roadmap graph [22].

In a direct comparison based on our own prior published work, the spanner algorithm presented in [22] (WSS) on average retains 20 to 30% of the edges needed by PRM$*$. The number of edges retained can drop to around 10% or even lower if a large enough stretch for the spanner is provided. At the same time, the average path quality for the WSS is 10 to 20% worse than that of PRM$*$, due to the loss of edges. The quality of paths does not change much even for a significantly large stretch. One advantage of the WSS is its runtime, which is 50 to 70% faster compared to PRM$*$, as many collision detections are not needed for the edges discarded by WSS online.

In conclusion, the WSS spanner algorithm, though faster, retains usually 10 times the data needed compared to the regression-based approach, and still outputs paths with worse quality.

V. Use case: high memory cost cell planners

One key advantage of using the regression complex approach is its ability to process the configuration space Q on a cell by cell basis, avoiding the need to store each cell planner for the entire complex. Once a cell C has its all-pairs distance information summarized via the regressor R_C and classifier K_C, all information specific to that cell planner can be discarded. Consider an instance of a motion planning problem where a PRM* is used as the cell planner. If PRM* must be used across the entire space, the entire roadmap must be maintained and searched across. With a regression complex, a PRM* is constructed in each cell, which is then summarized and discarded. When a query is run on the regression complex, cells for which the cell planner is required (see subsection III-C) can reconstruct the PRM* in that cell alone. Figure 7 depicts the regression complex run on a maze with a PRM* cell planner. Note that although Figure 7 shows PRM*s for each cell was relevant to the query, there is no need to keep more than one cell's cell planner in memory at any given time – once the path is found, the cell planner may be discarded. This approach allows motion planning for problems where planning over the entire configuration space requires too much memory to be computationally feasible. In the example shown by Figure 7, the regression complex had a maximum memory usage of $163,200$ floating points, whereas a PRM* of equivalent path quality required a maximum memory usage of $460,000$ floating points.



(7) Reconstructed cell planners for a regression complex on for a problem with an intricate configuration space.

VI. PLRC* Asymptotic Completeness and Optimality

The following proofs refer to a PLRC* is used to solve a motion planning problem (Q, q_s, q_t) with a robust solution $\sigma^* : [0,1] \mapsto Q_f$ with strong δ-clearance.

Let $\mathcal{C}$ denote the complex of regression cells generated by the PLRC*. Let $G_b = (V_b, E_b, w_b)$ be the query graph described in subsubsection III-C.1. Assume that each cell $C \in \mathcal{C}$ uses PLR with minimum cell edge size ϵ as regressor and classifier, and that boundary samples are placed in a grid with step size ϵ_s.

Because the cells in $\mathcal{C}$ cover all of Q, any path between configurations must trivially cross the boundaries of $\mathcal{C}$. We can define these crossing points as the values $b_1, \ldots, b_n$ of

σ^* which corresponds to a path $\pi_b = (\sigma^*(b_1), \ldots, \sigma^*(b_n))$ such that each $\sigma^*(b_i) \in B(\mathcal{C})$.

Lemma 1 (Convergence of boundary graph accuracy): Let π^* denote the shortest path from q_s to q_t on boundary graph G_b. As $\epsilon \to 0$ and $\epsilon_s \to 0$, the distance between each configuration $\pi^*[i]$ and $\pi_b[i]$ approaches 0.

Proof: Because the boundary sampling method samples cells with a sampling width of ϵ_s, any configuration $q \in B(\mathcal{C})$ is at most $dim * \epsilon_s$ away from a sample q_V, where dim is the dimensionality of the C-space Q. Thus as $\epsilon_s \to 0$, the distance from q to the nearest configuration $q_V \in V$ approaches 0. To show that the nearest q_V is on the path π^*, we show that the edge weights w_b match the distance function d at the limit of ϵ.

Given a value function $V(\cdot)$ with Lipschitz continuity factor κ, we know from Theorem 2 of [17] that for the approximation $L(\cdot)$ from the PLR, $|V(q) - L(q)| \leq \frac{5}{2}\kappa\epsilon\sqrt{n}\ \forall q \in Q$. Consider the regression R_C over the all pairs distance function d, which PLRC* uses to calculate the graph weights w_b. By the inequality above, as $\epsilon \to 0, |R_C(q_1, q_2) - d(q_1, q_2)| \to 0$ as long as q_1 and q_2 are correctly classified. The accuracy of K_C with respect to the connectivity function converges to 0 by the same argument.

Because w_b is a good approximate of d in the limit of ϵ and π^* is the shortest path on G, we know that if configurations are sampled close to each boundary waypoints q, the nearest configuration $q_V \in V$ must be on π^*. Therefore as $\epsilon \to 0$ and $\epsilon_s \to 0$, the distance between each configuration $\pi^*[i]$ and $\pi_b[i]$ approaches 0. ■ ■

Theorem 1 (Resolution completeness of LPRC):* LPRC* is resolution complete with respect to ϵ and ϵ_s.

Proof: Let σ denote the path produced by PLRC*. σ is constructed from the waypoints in π^* using the cell planner between each consecutive pair of configurations (q_i, q_{i+1}) on the path π_b. By lemma 1, as $\epsilon \to 0$ and $\epsilon_s \to 0$, a shortest path π_b on G_b must exist connecting q_s to q_t. Because the cell planner PRM* is guaranteed to return a feasible path at high enough resolution, σ must be feasible as long as each consecutive pair of configurations (q_i, q_{i+1}) is connectable. The feasibility of the path between each (q_i, q_{i+1}) depends on the error K_C, which as argued above, converges to 0 as $\epsilon \to 0$. Thus as ϵ and ϵ_s approach 0, PLRC* is guaranteed to return a feasible solution. ■ ■

Theorem 2 (Asymptotic optimality of PLRC):* RCRM is asymptotically optimal with respect to ϵ and ϵ_s.

Proof: Again, let σ denote the path produced by PLRC*. By Theorem 1, we know that the solution σ is feasible, and by lemma 1 the distance between corresponding boundary configurations $\pi^*[i]$ and $\pi_b[i]$ approaches 0. Given a sufficiently dense PRM*, the path returned by the cell planner is the optimal path between each waypoint on π_b. Since each configuration on π_b is arbitrarily close to π^*, the error of total path length converges to 0 with ϵ and ϵ_s. ■

VII. Future work

The primary contribution of this work is the notion of using spatial decomposition and regressions to construct a hybrid representation of a search space. We compared the performance of the approach against PRM* in various search spaces and provide a model-agnostic algorithm for regressing distances in search spaces for motion planning. We have tested alternative regression techniques, like XGBoost and neural networks, but not deeply.

Computing all-pairs distance using a dense PRM* and repeatedly running A* search is not an elegant approach. Although the regression complex can avoid constructing a dense graph for the entire search space since it only needs roadmaps in local cells, there is still a lot of computation that goes into constructing these local roadmaps. Setting the limit on the density of the PRM* is also unclear; denser roadmaps provide more optimal paths, but at the cost of construction time and memory. Our experiments navigated this tradeoff through trial and error, and finding the optimal limits on PRM construction will require further experimentation.

Similarly, the computational costs of refinement are significant and may pose a limit on the practical applications of this work. As mentioned in subsection I-B, alternative approaches using smooth approximations during the query phase like that employed by Field D* [10] might be used to reduce the costs of our preliminary approach.

References

[1] Devin Balkcom, Ajay Kannan, Yu-Han Lyu, Weifu Wang, and Yinan Zhang. "Metric cells: Towards complete search for optimal trajectories". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 4941–4948.

[2] Jérôme Barraquand and Jean-Claude Latombe. "Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles". In: *International Conference on Robotics and Automation*. Sacramento, CA, 1991, pp. 2328–2335.

[3] Mukunda Bharatheesha, Wouter Caarls, Wouter Jan Wolfslag, and Martijn Wisse. "Distance metric approximation for state-space RRTs using supervised learning". In: 2014, pp. 252–257.

[4] Joshua Bialkowski, Sertac Karaman, Michael W. Otte, and Emilio Frazzoli. "Efficient Collision Checking in Sampling-Based Motion Planning". In: 2012, pp. 365–380.

[5] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM. 2016, pp. 785–794.

[6] Hao-Tien Lewis Chiang, Jasmine Hsu, Marek Fiser, Lydia Tapia, and Aleksandra Faust. "RL-RRT: Kinodynamic Motion Planning via Learning Reachability Estimators from RL Policies". In: *CoRR* abs/1907.04799 (2019).

[7] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. "A Survey on Network Embedding". In: *CoRR* abs/1711.08752 (2017).

[8] Robin Deits and Russ Tedrake. "Computing Large Convex Regions of Obstacle-Free Space through Semidefinite Programming". In: *Workshop on the Algorithmic Foundations of Robotics (WAFR)*. Istanbul, Turkey, Aug. 2014.

[9] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. "PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-Based Planning". In: *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. IEEE, 2018, pp. 5113–5120.

[10] David Ferguson and Anthony (Tony) Stentz. *The Field D* Algorithm for Improved Path Planning and Replanning in Uniform and Non-Uniform Cost Environments*. Tech. rep. CMU-RI-TR-05-19. Pittsburgh, PA: Carnegie Mellon University, June 2005.

[11] Sertac Karaman and Emilio Frazzoli. "Sampling-based Algorithms for Optimal Motion Planning". In: *The International Journal of Robotics Research* 30(7), 846-894 (2011).

[12] Lydia Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark Overmars. "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces". In: *IEEE International Conference on Robotics and Automation*. 1996, pp. 566–580.

[13] Anthanasios Krontiis, Andrew Dobson, and Kostas Bekris. "Sparse roadmap spanners". In: *Proceedings of the workshop on the algorithmic foundations of robotics on Robotics, WAFR 2012* (2012).

[14] Tomás Lozano-Pérez and Michael A. Wesley. "An Algorithm for Planning Collision-free Paths Among Polyhedral Obstacles". In: *Commun. ACM* 22.10 (Oct. 1979), pp. 560–570. ISSN: 0001-0782. DOI: 10.1145/359156.359164. URL: http://doi.acm.org/10.1145/359156.359164.

[15] James D. Marble and Kostas E. Bekris. "Asymptotically Near-Optimal Planning With Probabilistic Roadmap Spanners". In: *IEEE Transactions on Robotics* 29.2 (2013), pp. 432–444.

[16] Rémi Munos, Leemon C. Baird, and Andrew W. Moore. "Gradient descent approaches to neural-net-based solutions of the Hamilton-Jacobi-Bellman equation". In: *IJCNN*. 1999.

[17] Josiah Putman, Lisa Oh, Luyang Zhao, Evan Honnold, Galen Brown, Weifu Wang, and Devin Balkcom. "Piecewise linear regressions for approximating distance metrics". In: *arXiv* (2020).

[18] D. Chris Rayner, Michael H. Bowling, and Nathan R. Sturtevant. "Euclidean Heuristic Optimization". In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. 2011.

[19] J. A. Reeds and L. A. Shepp. "Optimal paths for a car that goes both forwards and backwards." In: *Pacific J. Math.* 145.2 (1990), pp. 367–393. URL: https://projecteuclid.org:443/euclid.pjm/1102645450.

[20] T. Siméon, J.-P. Laumond, and C. Nissoux. "Visibility-based probabilistic roadmaps for motion planning". In: *Advanced Robotics* 14.6 (2000), pp. 477–493.

[21] Russ Tedrake, Ian R. Manchester, Mark M. Tobenkin, and John W. Roberts. "LQR-trees: Feedback Motion Planning via Sums-of-Squares Verification". In: *I. J. Robotics Res.* 29.8 (2010), pp. 1038–1052.

[22] Weifu Wang, Devin J. Balkcom, and Amit Chakrabarti. "A fast online spanner for roadmap construction". In: *I. J. Robotics Res.* 34.11 (2015), pp. 1418–1432. DOI: 10.1177/0278364915576491. URL: https://doi.org/10.1177/0278364915576491.