Learning Approximate Cost-to-Go Metrics To Improve Sampling-based Motion Planning

Yanbo Li Kostas E. Bekris

***Abstract*— Sampling-based planners have been shown to be effective in searching unexplored parts of a system's state space. Their desirable properties, however, depend on the availability of an appropriate metric, which is often difficult to be defined for some robots, such as non-holonomic and under-actuated ones. This paper investigates a methodology to approximate optimum cost-to-go metrics by employing an offline learning phase in an obstacle-free workspace. The proposed method densely samples a graph that approximates the connectivity properties of the state space. This graph can be used online to compute approximate distances between states using nearest neighbor queries and standard graph search algorithms, such as A*. Unfortunately, this process significantly increases the online cost of a sampling-based planner. This work then investigates ways for the computationally efficient utilization of the learned metric during the planner's online operation. One idea is to map the sampled states into a higher-dimensional Euclidean space through multi-dimensional scaling that retains the relative distances represented by the sampled graph. Simulations on a first-order car and on an illustrative example of an asymmetric state space indicate that the approach has merit and can lead into more effective planning.**

I. Introduction

Sampling-based motion planners [1], [2] are effective by utilizing properties for quickly exploring a moving system's state space. For instance, the RRT algorithm [3] automatically utilizes a Voronoi bias during the sampling process. The existence of such desirable properties, however, depends on the availability of an appropriate metric in the state space [4], [5]. Proper metrics are often difficult to be defined for important robotic systems, such as non-holonomic and under-actuated ones. The complication is that the computation of a proper metric, or a quasi-metric for systems where the path cost is not symmetric, often requires the solution to an optimal motion planning problem [6]. The worse the performance of the metric in accurately representing the optimal path cost, the worse the planner in covering the space. Similarly, the more difficult it becomes to solve hard planning instances.

Fig. 1 illustrates the challenge. It projects states sampled by RRT for a first-order car-like vehicle on the x and θ coordinates. Notice the gaps in coverage along the X axis for $\theta = \frac{\pi}{2}$ or $-\frac{\pi}{2}$ for a typical Euclidean metric. The holes correspond to the perpendicular orientation relative to the initial configuration along the X axis (initial configuration

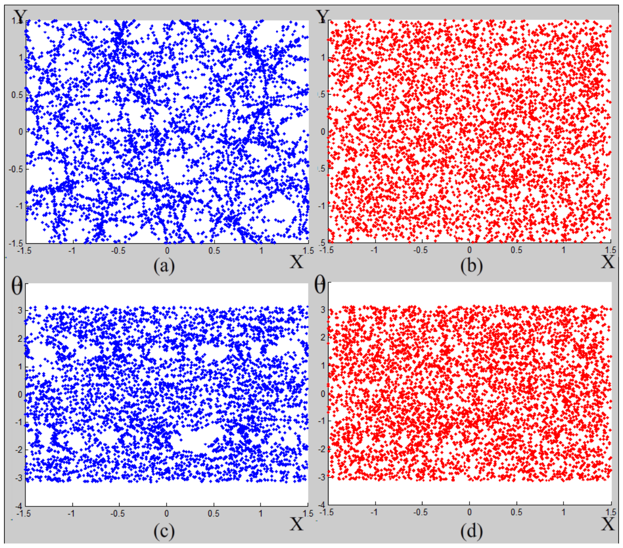


Fig. 1. States along the nodes of trees constructed with RRT for a first-order car-like system. The states are mapped to the x, y and x, θ projections, where x, y are Cartesian coordinates and θ is the car's orientation. (left) States sampled by RRT using a weighted Euclidean metric after 5K iterations. (right) RRT utilizing the learned metric after 5K iterations.

$0, 0, 0$). This indicates that RRT cannot easily extend states into this part of the state space using an ineffective metric. Notice, however, that once the algorithm is provided a better metric it is able to achieve uniform coverage.

This paper investigates a general method for automatically generating appropriate metrics for sampling-based planners. The idea is to utilize an offline process, where a large number of states are densely sampled in an obstacle-free environment. For each state, a large number of controls are extended to discover neighborhood relations between states. Two states are considered neighbors, if a control on the first state brings the system close to the second one. This can be viewed as computing a Probabilistic Roadmap [7] in the state-space of the system. The roadmap captures the connectivity properties of the state space due to constraints that are encoded in the system's forward propagation function. While this process introduces gaps [8], it becomes increasingly accurate as the number of states and controls increases, and the vicinity of a state decreases. Given the sampled states and their connectivity, a graph can be constructed and an approximate path cost between two sampled states can be computed using graph search, such as A*. Then the distance between two general states can be approximated by the path cost between their closest sampled states on the graph. The right side of Fig. 1 illustrates the improved coverage of RRT when it uses the result of the path cost on the graph as a metric.

This work is supported by NSF through project CNS 0932423. Any opinions and conclusions expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsor.

The authors are with the Computer Science and Engineering Department of the University of Nevada, Reno, 1664 N. Virginia St., MS 171, Reno, NV, 89557. Correspondence: bekris@cse.unr.edu

The above offline process, while computationally intensive, takes place only once for each system. Moreover, symmetries in the state-space can be utilized to reduce the number of states sampled during the offline process. Nevertheless, the approach also imposes a significant overhead during online planning. Both the nearest neighbor queries in RRT and the calls to the graph search algorithm become increasingly expensive as the number of sampled states and the connectivity of the offline graph increases, i.e., as the accuracy of the procedure improves.

Consequently, it is necessary to consider efficient alternatives to utilize the learned approximate metric. This work embeds the offline samples into a higher-dimensional Euclidean space in a way that retains their relative distances given their path costs on the offline graph. Multi-dimensional scaling (MDS) [9] can be employed for this purpose. It is a statistical method for exploring similarities between data, typically applied in non-linear dimensionality reduction, as in Isomap [10]. MDS can provide an embedding offline for every sampled state. Then instead of sampling a random state during the online phase, an offline computed state is randomly selected. Each node on the tree also stores the closest offline state. The distance between a tree node and a random state is computed by taking the Euclidean distance between the embeddings of the corresponding offline samples. Given the embeddings, it is also possible to employ a kd-tree to answer the nearest neighbor queries faster.

MDS introduces an additional level of approximation. Moreover, it imposes an overhead compared to computing a metric directly from the original states, because it requires finding and storing the closest offline-sampled state for each node of the tree. Nevertheless, the simulations provided in this paper show that the overhead is significantly smaller compared to the cost of the graph search and an improvement over standard RRT is still achieved. An important advantage is that the approach is very general. A complication arises for systems where the cost function between states is not symmetric, i.e., going from state x to y is not the same as going from y to x. The complication is that MDS operates on metric spaces, and requires a symmetric distance matrix. This paper describes how it is still possible to apply the proposed methodology even for this type of systems.

II. Related Work

Sampling-based planners, such as RRT [3], construct a tree data structure T in the state space $\mathcal{X}$ and employ heuristics to explore $\mathcal{X}$ evenly. During each iteration they select a state along the tree that is already connected to the start. Then they apply a control from the selected state. In particular, RRT randomly samples $x_{rand} \in \mathcal{X}$ and then finds the closest state to x_{rand} along the tree: $x_{near} \in T$. Given an appropriate metric, the algorithm provides a Voronoi bias, where the larger unexplored parts of $\mathcal{X}$ have higher probability of being explored. A commonly used metric is a weighted Euclidean distance given the coordinates in $\mathcal{X}$. Weights are employed to scale the relative significance of the coordinates. This metric works sufficiently well for holonomic problems. Metrics for holonomic problems have been studied in the past [11]. For systems with motion constraints, however, this approach causes problems, as the one displayed in Fig. 1, since it does not reflect the effects of constraints.

Many variations have been proposed that aim to decrease metric sensitivity. Some of them reduce the rate of failed expansions of a node [4], [12]. Others guide the tree expansion using local reachability information [13]. The sensitivity of RRT to the selected metric has also been discussed in the context of applications in bioinformatics [14]. An ideal metric is the optimal cost-to-go function [4], [6]. For certain systems, the optimal paths can be computed [15], [16], but this is not true in general. There is work on approximating the ideal metric for specific systems, such as a blimp by considering its dynamics [17]. A recent approach considers linearization for automatically computing metrics for kinodynamic systems [5].

This work provides a general method for approximating the cost-to-go metric that requires access only to a forward propagation function and is applicable to a wide variety of systems. This includes physically-simulated systems for which direct access to state update equations may not be available. There have been previous efforts to utilize dimensionality reduction tools, such as Principal Component Analysis (PCA), in motion planning or in structural bioinformatics [18]. The current authors have worked on utilizing PCA to learn offline the effects of dynamic constraints in the exploration performance of sampling-based planners and correct for them during the online process [19].

III. Approach

This section discusses first "symmetric systems", which have paths that can be inverted with the same cost. Thus, the distance matrix for a set of states is symmetric. Examples include simple point robots, holonomic robots, or a first-order car that can drive both forwards and backwards. The opposite is true for "asymmetric" systems, such as a second-order car, a pendulum or an acrobot.

A. Symmetric systems

Creating a densely sampled graph: The first step is to densely sample n states in an obstacle-free workspace. For

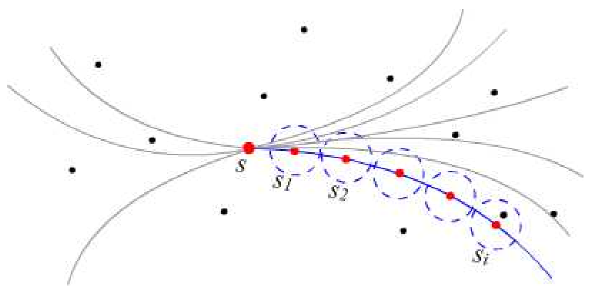


Fig. 2. Several controls are applied for each offline sample S. For each state S_i along the resulting trajectory, if there is another offline sample x within a predefined threshold, then S is considered a neighbor of x with the cost equal to the path cost of (S, S_i). This propagation terminates when the first neighbor is encountered.

each state, a c controls are applied, to compute connectivity between states. Fig. 2 shows an example. This step does introduce gaps but the graph is only intended to produce an estimate instead of the true metric. For symmetric systems, an undirected edge is created that stores the associated cost, such as time or energy to connect the two states. This will result in an undirected graph G for the n sampled states. As n and c increase, the graph better approximates the topology of the underlying manifold. The hope is that geodesic distances on the manifold are more accurate than Euclidean distances in the original space. Therefore, using a search algorithm on G, e.g., A^* or Dijkstra, it is possible to find an approximate optimal path between each two sampled states.

For computing the distance of two states x and y for `RRT`, the closest nodes x' and y' on G are first identified. Then the shortest path cost between x' and y' is used as the distance of x and y. Algorithm 1 summarizes the approach. The nearest offline sampled x'_i to the tree node x_i is found by employing the standard metric that would otherwise be used.

Algorithm 1 RRT using the precomputed graph G

```
$T$.init($x_0$)
for $i = 1$ $to$ $K$ (maximum # iterations) do
    $x_{rand} \leftarrow$ return a random $x \in G$
    $x_{near} \leftarrow$ GraphSearchNearestState($T, x_{rand}, G$)
    $x_{edge} \leftarrow$ Extend( $x_{near}, x_{new}$ )
    $T$.AddVertex($x_{edge}$)
    $T$.AddEdge( $x_{near}, x_{edge}$ )
```

Naively, the algorithm could perform a linear comparison using multiple graph searches for each node in the tree to the random state and return the one with the shortest path. However, all of these calls to A^* can be combined into a single one to save computation time. The steps are shown in Algorithm 2, which finds the shortest path from an already built tree to the newly randomly sampled state.

Algorithm 2 GraphSearchNearestState(T, x'_g, G)

```
$Fringe$ = empty
for $i = 1$ $to$ $m$ (# states in tree) do
    $x'_i \leftarrow$ ClosestStateInG($x_i \in T$)
    Fringe.Add($x'_i$)
for Commencing A* iterations do
    if $x'_g$ is reached then
        trace back and return $x'_i$
$x_{near} \leftarrow$ LookupStateInTree($x'_i$)
```

Embedding states to a Euclidean space: The computational cost of this A^*-based approach is far beyond the requirements of online computation. To speed-up this process, the idea is to seek a new representation of the graph, where distances between two states can be computed quickly using a new representation, i.e., an embedding, of these states. Euclidean distances based on these embeddings should represent as accurately as possible the path cost on graph G. This paper employs Multidimensional Scaling (`MDS`) [9] to map the shortest path cost on G to a Euclidean distance metric. An `MDS` algorithm starts with a matrix of item-to-item similarities, then assigns a location to each item in an N-dimensional space, where N is specified a priori.

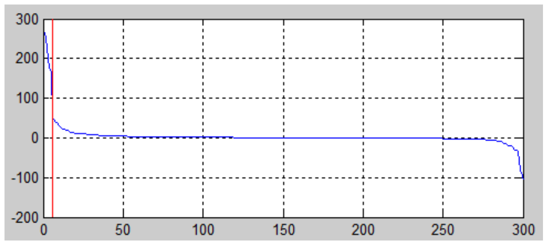
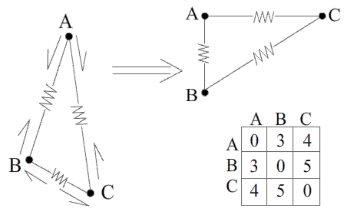


Fig. 3. Eigenvalues for a scalar product matrix of the distance matrix for 300 randomly sampled states. The data correspond to a first order car. The chosen dimension N is 6.

Determining the dimension N of the Euclidean space: A finite dimension N is guaranteed to exist for the space of embeddings, as long as the input distance matrix is symmetric and satisfies the triangular inequality. The offline graph G, however, introduces small gaps between the states and may violate the triangular inequality. If all the eigenvalues of the scalar product matrix of the distance matrix are positive, then this is possible [20]. Due to the existence of gaps in the graph, some negative eigenvalues do emerge. The parameter N is chosen in a way to eliminate this effect. Fig 3 shows an example of the resulting eigenvalues. The projection number N is determined by integrating the area below zero and offset the same area above zero from the right most side. The red line in Fig 3 shows the point where the positive area is equal to the negative one. The number of the left-most eigenvalues separated by the red line is the chosen dimension N.

Efficient computation of the embedding: For a large number of n sampled states, the distance matrix will be impossible to compute and store. Even so, for a finite dimension N, k times N distance constraints for each state are sufficient for `MDS` to produce useful embeddings. Empirically, k ranges between 10 and 20. Therefore, $kN \cdot n$ distances are required, which is linear to n, instead of n^2. This paper followed a spring-force model to compute the embeddings as in Fig. 4.

Online utilization of embeddings: After the offline learning, a table L of n entries is available. Each entry stores a pair of coordinates. The first coordinate is the original state. The second is the embedding in the Euclidean projection. During online planning, the distance of x and y is calculated by finding the closest sampled states x' and y' in the table (according to a standard metric). Then the approach looks up the new coordinates x'' and y'' which are embeddings for x' and y'. And then the Euclidean distance $||x'', y''||$ is employed to compute the distance between x and y.



	A	B	C
A	0	3	4
B	3	0	5
C	4	5	0

Fig. 4. A force $f = (||p_i - p_j||\text{-}d_{ij})e_{ij}$ is computed for states x_i and x_j, where d_{ij} is the provided distance, $|p_i - p_j|$ is the distance of the embeddings and e_{ij} is a normalization parameter. Given damping parameters and an annealing process, an optimization is used to find p_i, p_j.

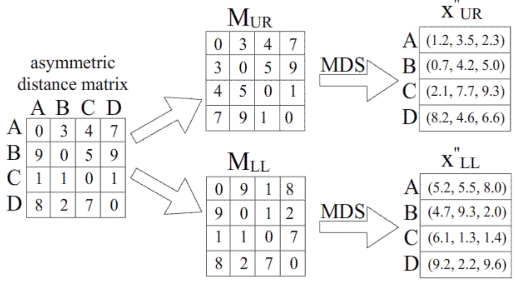


Fig. 5. An asymmetric distance matrix is split into two symmetric matrices M_{LL} and M_{UR}. MDS is applied on both matrices.

B. Asymmetric systems

In asymmetric systems, it is not possible to define a metric space but a quasi-metric one and the states can no longer be directly projected to a Euclidean space. However, it is possible to project the states into two artificial symmetric spaces. The states are assumed ordered based on a predefined criterion. This ordering can be arbitrary, as long as it is unique and can be recreated. For example, the states can be stored in ascending order based on a coordinate (e.g., along the X axis and for states with the same X coordinate, ordering them based on their Y coordinate, etc.). As before, the distance matrix is computed using A^* or Dijkstra's algorithm on the graph. The resulting asymmetric matrix is split into its lower-left and upper-right triangles as in Fig. 5. Two symmetric matrices are then defined, where each one of them copies values from either triangle of the original distance matrix and flips them diagonally to achieve a complete symmetric matrix. Then it is possible to run the same MDS process on the two matrices.

The resulting matrices may not directly satisfy the triangular inequality. However, there always exists an optimal constant c such that adding c to all the off-diagonal elements can eventually satisfy this property [9]. During the online distance measurement, one of the two Euclidean spaces can provide the appropriate approximation of the distance between the two states. Thus, each sampled state x' is assigned two new Euclidean coordinates x''_{LL} (from the lower-left matrix) and x''_{UR} (upper-right matrix).

During the online step, two states x and y are compared according to the order of the offline process for their closest sampled states x' and y'. If x' comes before y', then it is possible to use the coordinates x''_{UR} and y''_{UR}. Otherwise, use x''_{LL} and y''_{LL}. During each iteration of RRT, the tree should be extended towards the randomly sample. The correct embedding will come for some of the nodes from the LL symmetric matrix and for others from the UR matrix. The values acquired from the two matrices cannot be directly compared because the two sets of embeddings are derived from two separate MDS processes. The proposed algorithm returns the closest nodes from both spaces. Only one of the nodes is the true closest one according to the approximate metric. Consequently, the question is whether it does worth extending controls from one more node so as to use the metric. The experimental section investigates this trade-off.

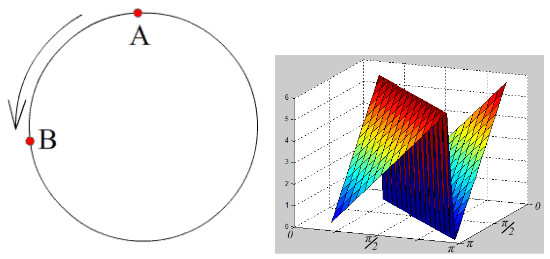


Fig. 6. (left) A one-dimensional circular space The robot can only travel counter-clock wise. (right) True distance matrix.

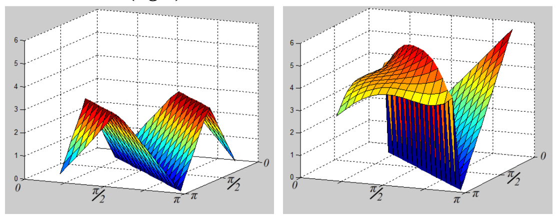


Fig. 7. (left) The distance matrix for the Euclidean metric. (right) The distance matrix learned from MDS.

IV. Simulations

A. An illustrative asymmetric space example

Consider a simple 1D circular space, such as a 2D point moving along a circle only in one direction, e.g., counter-clockwise, as in Fig. 6 (left). Fig. 6 (right) shows the graph of the distance matrix for the true metric. Fig. 7 (left) is the distance matrix of the Euclidean metric without knowledge of the asymmetry. The learned metric using MDS is shown on the right, using input points that are ordered according to their angles. In both pictures, the x and y axes are the angles of points on the circle. The vertical axis corresponds to the value of the elements in the distance matrix. The upper triangle is perfectly learned by the algorithm. The lower triangle does a better job in preserving the order of distances relative to the Euclidean distance.

Consider the numerical results shown in Table I acquired with the following procedure. A point along the circle was randomly selected as the target and 20 more points are randomly selected. Both the MDS-learned metric and the Euclidean metric are called to return the closest point to the target and evaluated for their accuracy relative to the perfect metric. The Euclidean metric has only 50% correct rate, since it will return the closest point in the unidirectional circle. Because the order of the points in the MDS-based approach matters, two results are provided for separate orderings. If the true closest neighbor is one of those two, then the MDS approach is considered successful. The MDS-based metric is able to partially approximate the topology. For the optimal

	True Neigh.	Same as Eucl.	Better than Eucl.	Worse than Eucl.	Dim required
points ordered based on their angle	9967	5055	4945	0	dim_{LL}=13 dim_{UR}=1
points ordered based on X coordinate	9318	4766	4918	316	dim_{LL}=8 dim_{UR}=8

TABLE I

Learned metric vs Euclidean metric.

ordering, based on the points' angle, the algorithm perfectly reconstructs half of the metric with one dimension (as shown from Fig. 7 (right)). For the upper triangle, it required 13 dimensions to achieve the best approximation. For the suboptimal ordering, the MDS has 93% chance to return the true closest state and 3% chance to return a state further away than the one return by the Euclidean metric. For the suboptimal choice, both symmetric spaces require 8 dimensions for the best possible approximation.

B. The approximate metric for a first-order car-like system

A benchmark employed for the symmetric case is a first-order car, which can go both forward and backward with the same ease. The car has 0.4m distance between its front and back wheels. The front wheel can steer between $[\pi/6, \pi/6]$. The car has a minimum turning radius of 0.693m. A graph G with 600k nodes is created for $x \in [-1.5, 1.5]$, $y \in [-1.5, 1.5]$ and $\theta \in [-\pi, \pi]$. For each state, 300 sampled controls are applied to calculate neighbors. This offline graph is computed using 20 cores in a parallel cluster. This process takes 5 hours on 20 cores, 2600MHz each. The optimal path cost on graph G is used as the new distance metric to grow an RRT tree (using Algorithm1) with 5000 iterations starting at $[0, 0, 0]$ (shown in Fig1 (right)). The result is compared against a weighted Euclidean distance metric (shown in Fig1(left)). Trees constructed by utilizing information from the graph G appear similar to a non-constrained system. Thus, using the new metric, the Voronoi bias exists in the new space where the metric was learned.

The standard RRT failed to cover areas where cars have $-\pi/2$ or $\pi/2$ orientation along the x axis. The results from the learned metric show that states from the tree are more uniformly distributed. Table II provides a coverage estimation for both methods. The state space is divided into cells. The density of states from the search tree within every cell is computed. The variance of the density of the cells evaluates the coverage. The lower the variance the more uniform the coverage is.

The previous experiment is encouraging but the computational cost is significantly higher than the straightforward RRT approach, since it is requires between 100 to 500 more time than the RRT, at least in an obstacle-free environment. In practical cases, collision checking and simulation could take significant portions of time costs. So the comparison could become beneficial for the algorithms described in this work as the cost of collision checking or simulation increase.

Figure 8 presents a comparison between the RRT that computes path costs on the graph G, two MDS-based RRTs with different number of embeddings and a weighted Euclidean metric RRT. The three algorithms were executed in an obstacle-free environment for up to 20K iterations. They were evaluated according to the following metric. A number of random states (10K in this case) are sampled to be potential goals. For each state, a small radius of 0.1 (normalized units) is considered as the goal region. If a tree samples a node in that region, then the corresponding goal is considered reached. This is a test to investigate how well the resulting tree covers the space. The figure shows how many of the goals were reached over an increasing number of iterations. This comparison indicates that on a per iteration basis, computing the path cost directly on the graph G provides the best coverage, while it has the highest time cost (approx. 500 times more than regular RRT). It requires about 43MB memory to store the graph G. The MDS-based RRT took almost the same computation time as the regular RRT but provided improved coverage. MDS-RRT with 150K embeddings has better coverage than that with 20k embeddings and requires 10MB. This indicates that memory can be traded for accuracy of the metric.

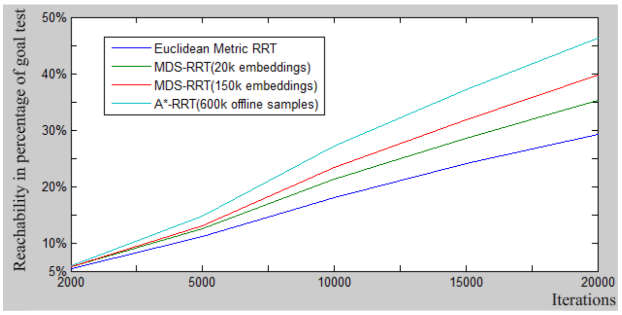


Fig. 8. Number of potential goals reached by sampling-based trees for different methods: Metric directly extracted from graph search on G (top), the middle two curves corresponds to the MDS-based metric with different numbers of embeddings, and a Euclidean metric RRT (bottom).

Table III provides the comparison in coverage between the learned metric and Euclidean metric for the same amount of time (20 seconds). A^*-RRT is significantly slower than the other two up to more than 500 times. The goal test for this method does not provide any useful result. The MDS-based RRT samples almost the same amount of states as an RRT with a weighted Euclidean metric during the same amount of time, while it achieves better coverage, since 30% more goal tests were successful.

Table IV explains the time spent for each phase during one iteration of RRT. The propagation step employs physics simulation and integration, which can take one more order of magnitude time than RRT computation. And the graph search directly on graph G requires another order of magnitude computation time. Apparently, computing the learned metric directly using the offline sampled graph G is not suitable for online problem solving.

Gird resolution	Euclidean RRT	A^*-based RRT
Density variance for 8,8,8	45.2	12.6
Density variance for 16,16,16	3.6	1.4

TABLE II
DENSITY VARIANCES FOR EUCLIDEAN RRT AND A^*-RRT.

	Euclidean RRT	MDS-based RRT	A^*-based RRT
Iterations	15564	15382	255
Potential goals reached	24.2%	31.5%	N/A

TABLE III
LEARNED METRIC VS EUCLIDEAN METRIC FOR THE SAME DURATION.

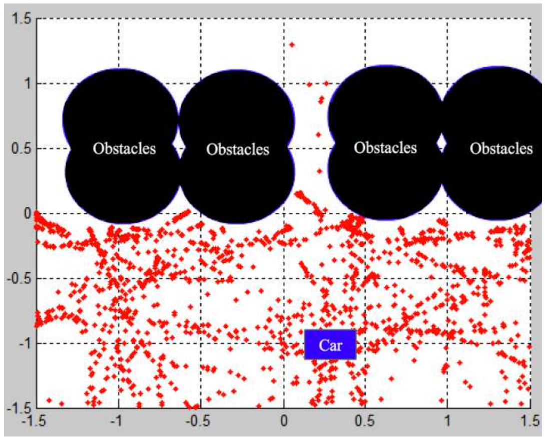


Fig. 9. The nodes sampled by the RRT algorithm using a weighted Euclidean metric after 2,000 iterations. There is a difficulty in passing through the narrow passage.

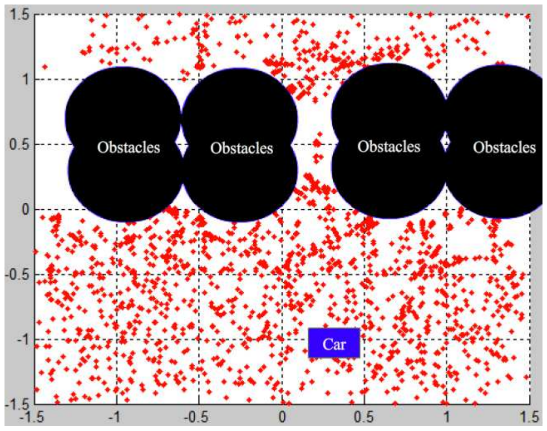


Fig. 10. The nodes sampled by the RRT algorithm using the path cost on graph G after 2,000 iterations. The algorithm was able to sample through the narrow passage.

C. Environment with obstacles

Consider the environment in Figs. 9 and 10, which contains a narrow passage. The Euclidean metric has a hard time guiding the RRT to explore the narrow passage, while the learned metric allowed its exploration for the same number of iterations (2000). Even though the metric is learned in an obstacle-free space, it can benefit the exploration of environments with obstacles.

V. Discussion

The paper proposes a general method for learning metrics in the state space of constrained moving systems. The first step is to densely sample an approximate connectivity graph in the state space to estimate the true optimal cost among many states. Utilizing this information during planning, however, has many computational drawbacks. To address this issue, the paper introduces a multi-dimensional scaling approach to compute higher-dimensional Euclidean embeddings for all the states sampled offline in a way that retains their relative distances in the computed graph. Simulations indicate that the computational cost of the MDS version of the metric is close to the standard RRT, even in the case where collision checking is not included in the cost. The accuracy of estimating the true cost-to-go metric is higher than the standard RRT but lower than directly utilizing the path cost from the graph constructed offline. Additional validation is required especially on systems with dynamic constraints, such as an inverted pendulum, an Acrobot robot and a second-order car-like system. Part of this process will be to investigate alternative methodologies to the existing way multi-dimensional scaling is performed so as to reduce the error introduced by the current procedure.

	Multi-start search on G	Propagation	Iter. basic RRT	Iter. MDS-RRT
Average time per iteration (s)	0.0775	0.0012	0.000078	0.000146

TABLE IV
Time cost for each phase during one iteration of RRT.

References

[1] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms and Implementations*. Boston: MIT Press, 2005.

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

[3] S. M. LaValle and J. J. Kuffner, “Randomized Kinodynamic Planning,” *IJRR*, vol. 20, no. 5, pp. 378–400, May 2001.

[4] P. Cheng and S. M. LaValle, “Reducing Metric Sensitivity in Randomized Trajectory Design,” in *Proc. of the IEEE/RSJ Intern. Conference on Intelligent Robots and Systems*, vol. 1, 2001, pp. 43–48.

[5] E. Glassman and R. Tedrake, “A Quadratic Regulator-based Heuristic for Rapidly Exploring State Space,” in *Proc. of the Intern. Conference on Robotics and Automation (ICRA)*, 2010.

[6] E. Frazzoli, M. A. Dahleh, and E. Feron, “Real-Time Motion Planning for Agile Autonomous Vehicles,” *Journal of Guidance, Control and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[7] L. Kavraki, P. Svestka, and J. C. Latombe, “Probabilistic Roadmaps for Path Planning in High-dimensional Configuration Spaces,” in *IEEE Trans. Robot. Automat.*, 1996, pp. 12(4):566–580.

[8] P. Cheng, E. Frazzoli, and S. M. LaValle, “Improving the Performance of sampling-based planners by using a symmetry-exploiting gap reduction algorithm,” in *ICRA*, 2004.

[9] I. Borg and P. Groenen, *Modern Multi-dimensional Scaling: Theory and Applications*. Springer-Verlag, 2005.

[10] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, “A Global Geometric Framework for Nonlinear Dimensionality Reduction,” *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[11] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, “Choosing Good Distance Metrics and Local Planners for Probabilistic Roadmap Methods,” in *Int. Conf. on Robotics and Automation*, 1998.

[12] A. Yershova, L. Jaillet, T. Simeon, and S. M. La Valle, “Dynamic-domain RRTs: Efficient Exploration by Controlling the Sampling Domain,” in *IEEE Intern. Conf. on Robotics and Automation*, 2005.

[13] A. Shkolnik, M. Walter, and R. Tedrake, “Reachability-guided Sampling for Planning under Differential Constraints,” in *IEEE Int. Conf. on Robotics and Automation*, 2009, pp. 2859–2865.

[14] J. Cortes, L. Jaillet, and T. Simeon, “Molecular Disassembly with RRT-like Algorithms,” in *IEEE ICRA*, 2007.

[15] L. Dubins, “On Curves of Minimal Length with a Constraint on Curvature and with Prescribed Initial and Terminal Positions and Tangents,” *Am. J. of Mathematics*, vol. 79, no. 1, pp. 497–516, 1957.

[16] J. Reeds and R. Shepp, “Optimal Paths for a Car That Goes Both Forwards and Backwards,” *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[17] J. Kim, J. Keller, and R. V. Kumar, “Design and Verification of Controllers for Airships,” in *Proc. of the 2003 IEEE/RSJ Inter. Conf. on Intelligent Robots and Systems (IROS)*, vol. 1, 2003, pp. 54–60.

[18] S. Dalibard and J.-P. Laumond, “Control of Probabilistic Diffusion in Motion Planning,” in *WAFR*, 2008.

[19] Y. Li and K. E. Bekris, “Balancing State-Space Coverage in Planning with Dynamics,” in *IEEE ICRA*, Anchorage, AK., May 2010.

[20] S. Messick and R. Abelson, *The Additive Constant Problem in Multidimensional Scaling*. Pyschometrika, Springer, 1956 March.