LEGO: Leveraging Experience in Roadmap Generation for Sampling-Based Planning

Rahul Kumar[1], Aditya Mandalika[2], Sanjiban Choudhury[2] and Siddhartha S. Srinivasa[2]

***Abstract*—We consider the problem of leveraging prior experience to generate roadmaps in sampling-based motion planning. A desirable roadmap is one that is sparse, allowing for speedy search, with nodes spread out at key locations such that a low-cost feasible path exists. An increasingly popular approach is to learn a distribution of nodes that would produce such a roadmap. Typically this is done by training a conditional variational auto-encoder (CVAE) on the prior dataset with the shortest paths as target input. While this is quite effective on many problems, we show it can fail in the face of complex obstacle configurations or mismatch between training and testing.**

We present an algorithm LEGO that addresses these issues by training the CVAE with target samples that satisfy two important criteria. Firstly, these samples belong only to *bottleneck* regions along near-optimal paths that are otherwise difficult-to-sample with a uniform sampler. Secondly, these samples are spread out across *diverse regions* to maximize the likelihood of a feasible path existing. We formally define these properties and prove performance guarantees for LEGO. We extensively evaluate LEGO on a range of planning problems, including robot arm planning, and report significant gains over both heuristics and learned baselines.

I. Introduction

We examine the problem of leveraging prior experience in sampling-based motion planning. In this framework, the continuous configuration space of a robot is sampled to construct a graph or *roadmap* [1, 2] where vertices represent robot configurations and edges represent potential movements of the robot. A shortest path algorithm [3] is then run to compute a path between any two vertices on the roadmap. The main challenge is to place a *small set of samples in key locations* such that the algorithm can find a high quality path with small computational effort as shown in Fig. 1b.

Typically, low dispersion samplers such as Halton sequences [4] are quite effective in uniformly covering the space and thus bounding the solution quality [5] (Fig. 1a). However, as they decrease dispersion uniformly in C-space, a narrow passage with δ clearance requires $O((\frac{1}{\delta})^d)$ samples to find a path. This motivates the need for biased sampling to *selectively densify* in regions where there might be a narrow passage [6–10]. These techniques are applicable across a wide range of domains and perform quite well in practice.

However, not all narrow passages are relevant to a given query. Biased sampling techniques, which do not have access

[1]Department of Computer Science, Indian Institute of Technology, Kharagpur {vernwalrahul}@iitkgp.ac.in

[2]Paul G. Allen School of Computer Science and Engineering, University of Washington {adityavk, sanjibac, siddh}@cs.uw.edu

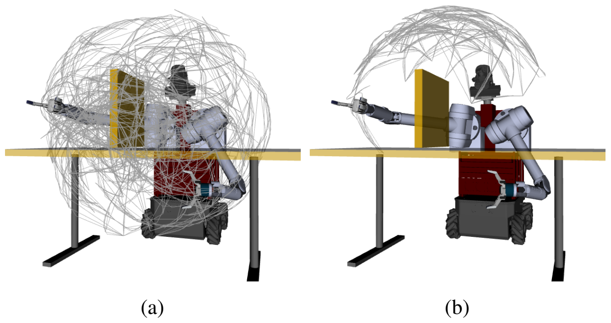


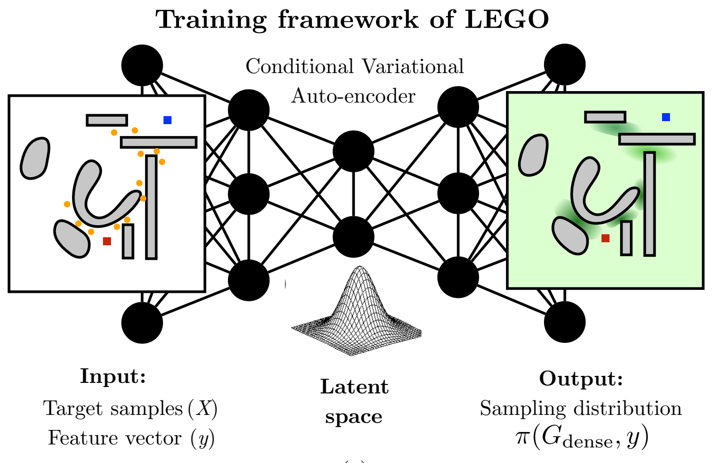
Fig. 1: Comparison of roadmaps generated from (a) uniform Halton sequence sampler and (b) from a generative model trained using LEGO. The task is to plan from the shown configuration over the table and obstacle to the other side. The graph is visualized by end effector traces of the edges.

to the likelihood of the optimal path passing through a region, can still end up dropping samples in more regions than necessary. Interestingly, the different environments that a robot operates in share a lot of structural similarity. Hence, we can use information extracted from planning on one such environment for deciding how to sample on another; we can *learn* sampling distributions using tools such as a conditional variational auto-encoder (CVAE). Ichter et al. [11] propose a useful approximation to train a learner to drop samples along the *predicted* shortest path: given a training dataset of worlds, compute shortest paths, and train a model to independently predict nodes belonging to the path. After all, the best a generative model can do, is to sample only along the true shortest path. However, this puts *all of the burden* on the learner. Any amount of prediction error, be it due to approximation or train-test mismatch, results in failure to find a feasible path.

We argue that a sampler, instead of trying to predict the shortest path, needs to only identify key regions to focus sampling at, and let the search algorithm determine the shortest path. Essentially, we ask the following question:

> How can we share the responsibility of finding the shortest path between the sampler and search

Our key insight is for the sampler to predict not the shortest path, but samples that possess two main characteristics. First, we only need to predict samples in bottleneck regions. These are regions containing near-optimal paths, but are difficult for a uniform sampler to reach. Secondly, we need diveristy amongst samples. Train-test mismatch is common and to be robust against it we need to sample nodes belonging to a diverse set of alternate paths. The search algorithm can then operate on a sparse graph containing useful but diverse samples to compute the shortest path.



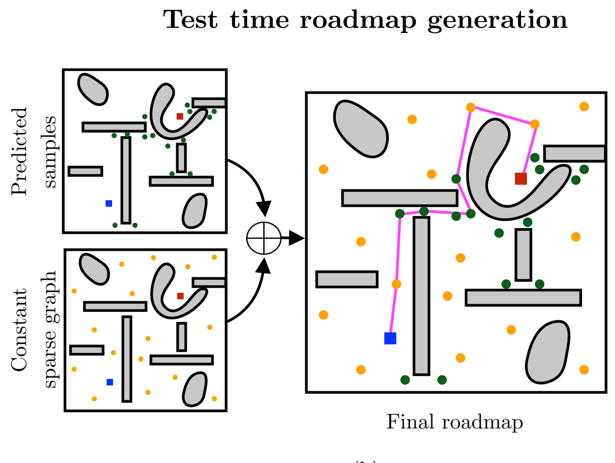


Fig. 2: The LEGO framework for training a CVAE to predict a roadmap. (a) The training process for learning a generative sampling distribution using a CVAE. The input is a pair of candidate samples and feature vector. (b) At test time, the model is sampled to get vertices which are then composed with a constant sparse graph to get a final roadmap.

We present an algorithmic framework, **L**everaging **E**xperience with **G**raph **O**racles (LEGO) summarized in Fig. 2, for training a CVAE on a prior database of worlds to learn a generative model that can be used for roadmap construction. During training (Fig. 2a), LEGO processes a uniform dense graph to identify a sparse subset of vertices. These vertices are a *diverse* set of *bottleneck* nodes through which a near-optimal path must pass. These are then fed into a CVAE [12] to learn a generative model. At test time (Fig. 2b), the model is sampled to get a set of vertices which is additionally composed with a sparse uniform graph to get a final roadmap. This roadmap is then used by the search algorithm to find the shortest path.

We make the following contributions:

1) A framework for training a CVAE to predict a roadmap with different target inputs. We identify two main shortcomings of the state-of-the-art [11] that uses the shortest path as the target input - failures in approximation, and failures due to train-test mismatch (Section IV).
2) LEGO, an algorithm that tackles both of these issues. It first generates multiple diverse shortest paths, and then extracts bottleneck nodes along such paths to use as the target input for the CVAE (Section V).
3) Show that LEGO outperforms several learning and heuristic sampling baselines on a set of $\mathbb{R}^2$,$\mathbb{R}^5$, $\mathbb{R}^7$, $\mathbb{R}^8$ and $\mathbb{R}^9$ problems. In particular, we show that it is robust to changes in training and test distribution (Section VI).

II. Related Work

The seminal work of Hsu et al. [13] provides a crisp analysis of the shortcomings of uniform sampling techniques in the presence of artifacts such as narrow passages. This has led to a plethora of non-uniform sampling approaches that densify selectively [6–10].

Adaptive sampling in the context of roadmaps aims to exploit structure of the environment to place samples in promising areas. A number of works exploited structure of the workspace to achieve this. While some of them attempt to sample between regions of collision to identify narrow passages [6, 14–18], others sample near or on the obstacles [19, 20]. There are approaches that divide the configuration space into regions and either select different region-specific planning strategies [21] or use entropy of samples in a particular region to refine sampling [22]. Other methods try to model the free space to speed up planning [23–25]. While these techniques are quite successful in a large set of problems, they can place samples in regions where an optimal path is unlikely to traverse.

A different class of solutions look at adapting sampling distributions online during the planning cycle. This requires a trade-off between exploration of the configuration space and exploitation of the current best solution. Preliminary approaches define a utility function to do so [26, 27] or use online learning [10]; however these are not amenable to using priors. Diankov and Kuffner [28] employs statistical techniques to sample around a search tree. Zucker et al. [29], Kuo et al. [30] formalize sampling as a model-free reinforcement learning problem and learn a parametric distribution. Since these problems are non i.i.d learning problems, they do require interactive learning and do not enjoy the strong guarantees of supervised learning.

There has been a lot of recent effort on finding low dimensional structure in planning [31]. In particular, generative modeling tools like variational autoencoders [32] have been used to great success [33–37]. We base our work on Ichter et al. [11] where a CVAE is trained to learn the shortest path distribution.

III. Problem Formulation

Given a database of prior worlds, the overall goal is to learn a policy that predicts a roadmap which in turn is used by a search algorithm to efficiently compute a high quality feasible path. Let $\mathcal{X}$ denote a $d-$dimensional configuration space. Let $\mathcal{X}_{\text{obs}}$ be the portion in collision and $\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obs}}$ denote the free space. Let a path $\xi : [0, 1] \rightarrow \mathcal{X}$ be a continuous mapping from index to configurations. A path ξ is said to be collision free if $\xi(\tau) \in \mathcal{X}_{\text{free}}$ for all $\tau \in [0,1]$. Let $c(\xi)$ be a cost functional that maps ξ to a bounded non-negative cost $[0, c_{\max}]$. Moreover, we set $c(\emptyset) = c_{\max}$. We define a *motion planning problem* $\Lambda = \{x_s, x_g, \mathcal{X}_{\text{free}}\}$ as a tuple of start configuration $x_s \in \mathcal{X}_{\text{free}}$, goal configuration $x_g \in \mathcal{X}_{\text{free}}$ and free space $\mathcal{X}_{\text{free}}$. Given a problem, a path ξ is said to be *feasible* if it is collision free, $\xi(0) = x_s$ and $\xi(1) = x_g$. Let Ξ_{feas} denote the set of all feasible paths. We wish to solve the *optimal* motion planning problem by finding a feasible path ξ^* that minimizes the cost functional $c(.)$, i.e. $c(\xi^*) = \inf_{\xi \in \Xi_{\text{feas}}} c(\xi)$.

We now embed the problem on a graph $G = (V, E)$ such that each vertex $v \in V$ is an element of $v \in \mathcal{X}$. The graph follows a connectivity rule expressed as an indicator function $\texttt{Link} : \mathcal{X} \times \mathcal{X} \to \{0, 1\}$ to denote if two configurations should have an edge[1]. The weight of an edge $c(u, v)$ is the cost of traversing the edge. We reuse ξ to denote a path on the graph.

We introduce a couple graph operations. Let $|G|$ denote the cardinality of the graph, i.e. the size of $|V|$[2]. We use the notation $G \overset{+}{\leftarrow} X$ to compactly denote insertion of a new set of vertices X, i.e. $V \leftarrow V \cup X$, $E \leftarrow E \cup \{(u, v) \mid u \in X, \texttt{Link}\,(u, v) = 1\}$.

A graph search algorithm $\textsc{Alg}$ is given a graph G and a planning problem Λ. First, it adds the start goal pair to the graph, i.e $G' = G \overset{+}{\leftarrow} \{x_s, x_g\}$. It then collision checks edges against $\mathcal{X}_{\text{free}}$ till it finds the shortest feasible path ξ^* which is then returned. Hence, the cost of such a path can be found by evaluating $c(\textsc{Alg}(G, \Lambda))$. If $\textsc{Alg}$ is unable to find any feasible path, it returns $\emptyset$ which corresponds to $c_{\max}$.

Definition 1 (Dense Graph)**.** *We assume we have a* dense graph $G_{\text{dense}} = (V_{\text{dense}}, E_{\text{dense}})$ *that is sufficiently large to connect the space i.e. for any plausible planning problem, it contains a sufficiently low cost feasible path.*

Henceforth, we care about competing with G_{dense}. We reiterate that searching this graph, $\textsc{Alg}(G_{\text{dense}}, \Lambda)$, is too computationally expensive to perform online.

We wish to learn a mapping from features extracted from the problem to a sparse subgraph of G_{dense}. Let $y \in \mathbb{R}^m$ be a feature representation of the planning problem. Let $\pi(G_{\text{dense}}, y)$ be a *subgraph predictor oracle* that maps the feature vector to a subgraph $G \subset G_{\text{dense}}$, such that $|G| \leq N$. We wish to solve the following optimization problem:

Problem 1 (Optimal Subgraph Prediction)**.** *Given a joint distribution* $P(\Lambda, y)$ *of features and problems, and a dense graph* G_{dense}*, compute a subgraph predictor oracle* π *that minimizes the ratio of the cost of the shortest feasible path in the subgraph to the dense graph:*

$$\pi^* = \underset{\pi \in \Pi}{\arg\min}\ \mathbb{E}_{(\Lambda, y) \sim P(\Lambda, y)} \left[\frac{c(\textsc{Alg}(\pi(G_{\text{dense}}, y), \Lambda))}{c(\textsc{Alg}(G_{\text{dense}}, \Lambda))} \right] \quad (1)$$

[1]Note this does not involve collision checking. We consider undirected graphs for simplicity. However, it easily extends to directed graphs.

[2]Alternatively we can also use the size of $|E|$

IV. Framework for Predicting Roadmaps

We now present a framework for training graph predicting oracles as shown in Fig. 2. The framework applies three main approximations. First, instead of predicting a subgraph $G \subset G_{\text{dense}}$, we train a Conditional Variational Auto-encoder (CVAE) [38] to approximate a sampling distribution of nodes $x \in \mathcal{X}$ in continuous space. Second, instead of solving a structured prediction problem, we learn an i.i.d sampler that will be invoked repeatedly to get a set of vertices. These vertices are then used to construct a graph. Thirdly, we compose the sampled graph with a *constant sparse graph* $G_{\text{sparse}} \subset G_{\text{dense}}, |G_{\text{sparse}}| \leq N$. This ensures that the final predicted graph has some minimal coverage. Refer to [39] for details.

A. General train and test procedure

a) Training Procedure: At train time Fig. 2(a), we follow four steps. First, we load a pair of planning problem Λ_i and feature y_i. Second, we obtain the target nodes $X_i = \textsc{ExtractNodes}(\Lambda_i, G_{\text{dense}})$. We feed the dataset $\mathcal{D} = \{X_i, y_i\}_{i=1}^{D}$ as input to CVAE. Finally, we train the CVAE and return the learned decoder $p_\theta(x|y, z)$.

b) Testing Procedure: At test time Fig. 2(b), we follow three steps. First, we extract feature vector y from planning problem Λ. We then sample a set V of N nodes using decoder $p_\theta(x|y, z)$. Finally, we create a composed graph $G \leftarrow G_{\text{sparse}} \oplus V$.

The focus of this work is to examine the function $X = \textsc{ExtractNodes}(\Lambda, G_{\text{dense}})$. We ask the question:

What is a good input X to provide to the CVAE?

B. The ShortestPath (Ichter et al. [11]) procedure

We briefly examine the scheme of training to predict nodes on the shortest path[11] as shown in Fig. 3. Formally, let $X_{\text{sp}} = \textsc{ShortestPath}(\Lambda, G_{\text{dense}})$ denote the set of nodes belonging to the shortest path returned by $\textsc{Alg}(G_{\text{dense}}, \Lambda)$. The rationality for this scheme is that the distribution of states belonging to the shortest path might lie on a manifold that can be captured by the latent space of the CVAE. If *prediction is perfect*, then indeed this is the optimal solution to Problem. 1.

After extensive evaluations of the ShortestPath scheme, we were able to identify two modes of vulnerabilities:

1) *Failure to route through gaps*: Fig. 3(b) shows the output of the CVAE when there is a gap through which the search has to route to get to the goal. The model gets stuck in a poor local minimum between linearly interpolating start-goal and routing through the gap since the network is not expressive enough to map the feature vector to such a path. This is tantamount to burdening the sampler to solve the planning problem.
2) *Presence of unexpected obstacles in test data*: Fig. 3(c) shows the output of the CVAE when there are small,

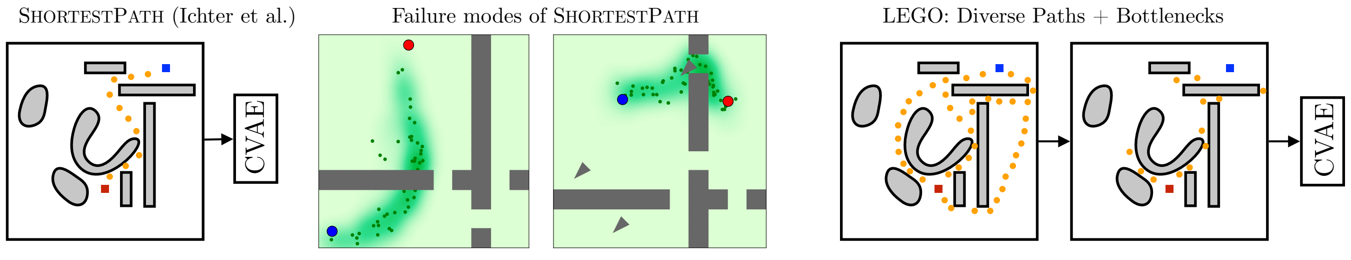

SHORTESTPATH (Ichter et al.)
Failure modes of SHORTESTPATH
LEGO: Diverse Paths + Bottlenecks
CVAE
CVAE
(a)
(b)
(c)
(d)

Fig. 3: (a) The training input generated by SHORTESTPATH. (b) Failure of SHORTESTPATH model to route through gaps (c) Another failure of the model due to unexpected obstacles (d) The training input generated by LEGO. Diverse shortest paths are generated followed by extraction of bottleneck nodes.

Algorithm 1: LEGO

```
Input  : Planning problem $\Lambda$,
         Dense graph $G_{\text{dense}}$, Sparse graph $G_{\text{sparse}}$
Output : LEGO nodes $V_{\text{lego}}$
1 $V_{\text{lego}} \leftarrow \emptyset$;
2 $\Xi_{\text{div}} \leftarrow$ DIVERSEPATHSET$(\Lambda, G_{\text{dense}})$;
3 for $\xi \in \Xi_{\text{div}}$ do
4     $V_{\text{bn}} \leftarrow$ BOTTLENECKNODE$(\Lambda, \xi, G_{\text{sparse}})$;
5     $V_{\text{lego}} \leftarrow V_{\text{lego}} \cup V_{\text{bn}}$
6 return $V_{\text{lego}}$;
```

Algorithm 2: BOTTLENECKNODE

```
Input  : Planning problem $\Lambda$,
         Dense path $\xi_{\text{dense}}$, Sparse graph $G_{\text{sparse}}$
Output : Bottleneck nodes $V_{\text{bn}}$
1 $G_{\text{inf}} \leftarrow G_{\text{sparse}} \oplus \xi_{\text{dense}}$, $\eta \leftarrow 1$ ;
2 while $c(\text{ALG}(G_{\text{inf}}, \Lambda)) \leq (1+\epsilon)c(\xi_{\text{dense}})$ do
3     $\eta \leftarrow \eta + \delta\eta$ ;
4     Inflate by $\eta$ the weight of edges $e \in E_{\text{inf}} \setminus E_{\text{sparse}}$ ;
5 $V_{\text{bn}} \leftarrow \xi_{\text{dense}} \cap \text{ALG}(G_{\text{inf}}, \Lambda)$ ;
6 return $V_{\text{bn}}$;
```

unexpected obstacles in the test data which were not present in the training data. The learned distribution samples over this obstacle as it only predicts what it thinks is the shortest path. Even if we were to have such examples in the training data, unless the feature extractor detects such obstacles, the problem remains.

V. Approach

In this section, we present LEGO (Leveraging Experience with Graph Oracles), an algorithm to produce the training input for the CVAE. We do so by tackling head-on the challenges identified in in Section IV. Firstly, we recognize that it is often too difficult for the learner to directly predict the shortest path. Instead, we train it to predict only *bottleneck nodes* that can assist the underlying search in finding a near-optimal solution. Secondly, the test data may contain unexpected obstacles. We safeguard against this by training the learner to predict a *diverse set of paths* with the hope that at-least one of them is feasible.

Algorithm 1 describes a high level pseudo-code of LEGO. We first find a set of diverse paths on the dense graph (Line 2). We then iterate over each path and extract bottleneck nodes for this path (Line 4). These are added to the set of nodes to be returned. These nodes are then used as input to the CVAE to learn a sampling distribution (Section IV). This is illustrated in Fig. 3. We now describe how bottleneck nodes and diverse paths are computed. The detailed algorithms and proofs can be found in the supplementary [39].

A. Bottleneck Nodes

We begin by noting that G_{sparse} has a sparse but uniform coverage over the entire configuration space. Hence, the learner only has to contribute a critical set of nodes that allow G_{sparse} to represent paths that are near-optimal with respect to a path ξ_{dense} generated from a dense graph. We call these *bottleneck nodes*. The name indicates that these are difficult-to-sample nodes, absent in the sparse graph, which are otherwise required to approximate ξ_{dense}. We define $X_{\text{bn}} = \text{BOTTLENECKNODE}(\Lambda, G_{\text{dense}})$ as:

Definition 2 (Bottleneck Nodes)**.** *Given a path on a dense graph ξ_{dense}, find the smallest set of nodes which in conjunction*[3] *with a sparse subgraph G_{sparse} contains a near-optimal path, i.e.*

$$\begin{aligned} \underset{V \subset \xi_{\text{dense}}}{\arg\min} \quad & |V| \\ \textit{s.t.} \quad & c(\text{ALG}(G_{\text{sparse}} \oplus V, \Lambda)) \leq (1+\epsilon)c(\xi_{\text{dense}}) \end{aligned} \tag{2}$$

The optimization Section 2 is combinatorially hard. We present an approximate solution in Algorithm 2. We begin by creating a compposed graph $G_{\text{sparse}} \oplus \xi_{\text{dense}}$ (Line 1). We inflate weights of newly added edges by η (Line 3). The idea is to disincentivize the search from using any of the newly added edges. This inflation factor is increased till a near-optimal path can no longer be found (Lines 2-4). At this point, the additional vertices that the shortest path on this inflated path pass through are essential to achieve near-optimality, i.e. are the bottleneck nodes. This is formalized by the following guarantee:

Proposition 1 (Bounded bottleneck edge weights)**.** *Let $E_{\text{bn}} \leftarrow \xi^*_{\text{inf}} \setminus E_{\text{sparse}}$ be the chosen bottleneck edges, E^*_{bn} be the optimal bottleneck edges and ξ^*_{dense} be the optimal path on G_{dense}.*

$$\sum_{e_i \in E_{\text{bn}}} c(e_i) \leq \sum_{e_i \in E^*_{\text{bn}}} c(e_i) + \frac{(1+\epsilon)c(\xi^*_{\text{dense}})}{\eta} \tag{3}$$

[3]Here $G \oplus V'$ represents a merge operation, i.e. $V \leftarrow V \cup V'$, $E \leftarrow E \cup \{(u,v) \mid u \in V', (u,v) \in E_{\text{dense}}\}$.

Algorithm 3: DIVERSEPATHSET

```
Input  : Planning problem $\Lambda$,
         Dense graph $G_{\text{dense}}$, Sparse graph $G_{\text{sparse}}$
Output : Diverse pathset $\Xi_{\text{div}}$
1 $\Xi \leftarrow \text{ALG}^{L}(G_{\text{dense}}, \Lambda)$ ;        ▷ L-shortest paths
2 for $i = 1, \cdots, k$ do
3     $E_i \leftarrow \emptyset$;
4     while $|E_i| < \ell$ do
5         Add the next $\ell - |E_i|$ paths from $\Xi$ to $\Xi$;
6         $E_i \leftarrow \text{SETCOVER}(\Xi_{\text{inv}})$;
7     Add the next path from $\Xi$ to $\Xi_{\text{div}}$;
8 return $\Xi_{\text{div}}$;
```

Fig. 4(a,b) illustrate the samples generated by SHORTESTPATH and BOTTLENECKNODE respectively, and the successful routing through narrow passages using samples generated by BOTTLENECKNODE.

B. Diverse PathSet

In this training scheme, we try to ensure the roadmap is *robust* to various sources of errors such as the test world contains unexpected obstacles or that the feature vector does not catpure sufficient details of the world. One antidote to this process is *diversity* of samples. Specifically, we want the roadmap to have enough diversity such that if the predicted shortest path is in fact infeasible, there are low cost alternates.

We set this up as a two player game between a planner and an adversary. The role of the adversary is to invalidate as many shortest paths on the dense graph G_{dense} as possible with a fixed budget of edges that it is allowed to invalidate. The role of the planner is to find the shortest feasible path on the invalidated graph and add this to the set of diverse paths Ξ_{div}. The function $X_{\text{div}} = \text{DIVERSEPATHSET}(\Lambda, G_{\text{dense}})$ then returns nodes belonging to Ξ_{div}. We formalize this as:

Definition 3 (Diverse PathSet)**.** *We begin with a graph $G^0 = G_{\text{dense}}$. At each round i of the game, the adversary chooses a set of edges to invalidate:*

$$E_i^* = \underset{E_i \subset E, |E_i| \leq \ell}{\arg\max}\; c(\text{ALG}(G^{i-1} \ominus E_i, \Lambda)) \tag{4}$$

and the graph is updated $G^i = G^{i-1} \ominus E_i^*$. The planner choose the shortest path $\xi_i = \text{ALG}(G^i, \Lambda))$ which is added to the set of diverse paths Ξ_{div}.

The optimization problem (4) is similar to a set cover problem (NP-Hard [40]) where the goal is to select edges to cover (invalidate) as many paths as possible. If we knew the exact set of paths to cover, it is well known that a greedy algorithm will choose a near-optimal set of edges [40]. We have the inverse problem - we do not know how many consecutive shortest paths can be covered with a budget of ℓ edges.

Algorithm 3 describes the procedure. We first compute a large set of shortest feasible paths (Line 1). The adversary successive paths to eliminate till it runs out of budget (Line 5). It then runs greedy set cover on these paths to see if they can be eliminated with a smaller budget (Line 6). If so, the

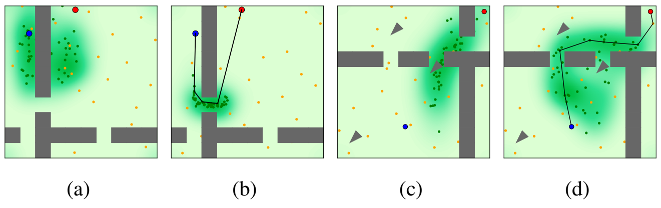


Fig. 4: (a) and (b) compare the samples generated by SHORTESTPATH and BOTTLENECKNODE respectively. (c) and (d) compare the samples generated by SHORTESTPATH and DIVERSEPATHSET respectively. In both instances, SHORTESTPATH fails to find a solution.

process continues till the set cover exhausts the budget. Hence at termination, we ensure:

Proposition 2 (Near-optimal Invalidated EdgeSet)**.** *Let Ξ_{inv} be the contiguous set of shortest paths invalidated by Algorithm 3 using a budget of ℓ. Let ℓ^* be the size of the optimal set of edges that could have invalidated Ξ_{inv}.*

$$\ell \leq (1 + \log |\Xi_{\text{inv}}|)\ell^* \tag{5}$$

Fig. 4(c,d) illustrate the samples generated by SHORTESTPATH and DIVERSEPATHSET respectively, and the robustness to unexpected obstacles exhibited by samples generated by DIVERSEPATHSET.

VI. EXPERIMENTAL RESULTS

In this section we evaluate the performance of LEGO on various problem domains and compare it against other samplers. We consider samplers that do not assume offline computation or learning such as Medial-Axis PRM (MAPRM) [6, 14], Randomized Bridge Sampler (RBB) [15], Workspace Importance Sampler (WIS) [16], a Gaussian sampler, GAUSSIAN [20], and a uniform Halton sequence sampler, HALTON [4]. Additionally, we also compare our framework against the state-of-the-art learned sampler SHORTESTPATH [35] upon which our work is based.

a) Evaluation Procedure: For a given sampler and a planning problem, we invoke the sampler to generate a fixed number of samples. We then evaluate the performance of the samplers on three metrics: a) sampling time b) success rate in solving shortest path problem and c) the quality of the solution obtained, on the graph constructed with the generated samples.

b) Problem Domains: To evaluate the samplers, we consider a spectrum of problem domains. The $\mathbb{R}^2$ problems have random rectilinear walls with random narrow passages (Fig. 6(a)). These passages can be small, medium or large in width. The n-link arms are a set of n line-segments fixed to a base moving in an uniform obstacle field (Fig. 6(b)). The n-link snakes are arms with a free base moving through random rectilinear walls with passages (Fig. 6(c)). Finally, the manipulator problem has a 7DoF robot arm [41] manipulating a stick in an environment with varying clutter (Fig. 6(d)). Two variants are considered - constrained ($\mathbb{R}^7$), when the stick is welded to the hand, and unconstrained, when the stick can slide along the hand ($\mathbb{R}^8$).

TABLE I: Average time (sec.) by sampling algorithms to generate 200 samples over 100 planning problems

	Non-Learned Samplers					Learned Samplers	
	HALTON	MAPRM	RBB	GAUSSIAN	WIS	SHORTESTPATH	LEGO
Point Robot (2D)	0.0036	0.53	0.22	0.02	0.25	0.006	0.006
N-link Arm (3D)	0.0058	–	23.96	1.95	0.36	0.016	0.016
N-link Arm (7D)	0.0071	–	37.24	3.77	1.12	0.017	0.017
Snake Robot (5D)	0.0069	39.56	142.21	3.43	0.54	0.013	0.013
Snake Robot (9D)	0.0074	40.01	180.43	8.71	2.11	0.017	0.017
Manipulator (8D)	0.01	–	–	3.33	–	0.018	0.018

TABLE II: Success Rates of different algorithms on 100 trials over different datasets (reported with a 95% C.I.)

	Non-Learned Samplers					Learned Samplers	
	HALTON	MAPRM	RBB	GAUSSIAN	WIS	SHORTESTPATH	LEGO
2D Point Robot Planning							
2D Large (easy)	0.73 ± 0.08	0.73 ± 0.08	0.74 ± 0.09	0.65 ± 0.09	0.78 ± 0.08	0.86 ± 0.07	$\mathbf{0.97 \pm 0.03}$
2D Medium	0.48 ± 0.08	0.63 ± 0.09	0.61 ± 0.09	0.48 ± 0.10	0.63 ± 0.09	0.69 ± 0.09	$\mathbf{0.89 \pm 0.06}$
2D Small (hard)	0.36 ± 0.09	0.53 ± 0.09	0.48 ± 0.08	0.32 ± 0.09	0.52 ± 0.09	0.59 ± 0.09	$\mathbf{0.83 \pm 0.07}$
N-Link Arm							
3D	0.39 ± 0.09	–	0.54 ± 0.09	0.46 ± 0.10	0.52 ± 0.10	0.61 ± 0.09	$\mathbf{0.74 \pm 0.08}$
7D	0.29 ± 0.09	–	0.46 ± 0.09	0.41 ± 0.09	0.46 ± 0.09	0.57 ± 0.10	$\mathbf{0.71 \pm 0.08}$
N-Link Snake Robot							
5D	0.41 ± 0.09	0.42 ± 0.09	0.48 ± 0.10	0.41 ± 0.09	0.50 ± 0.10	0.77 ± 0.08	$\mathbf{0.84 \pm 0.07}$
9D	0.49 ± 0.09	0.45 ± 0.09	0.52 ± 0.10	0.51 ± 0.10	0.53 ± 0.09	0.82 ± 0.07	$\mathbf{0.86 \pm 0.07}$
Manipulator Arm Planning							
Unconstrained (8D)	0.24 ± 0.09	–	–	–	–	0.81 ± 0.08	$\mathbf{0.82 \pm 0.07}$
Constrained (8D)	0.09 ± 0.05	–	–	–	–	0.58 ± 0.09	$\mathbf{0.70 \pm 0.09}$

c) Experiment Details: For the learned samplers SHORTESTPATH and LEGO, we use 4000 training worlds and 100 test worlds. Dense graph is an $r-$disc Halton graph [5]: 2000 vertices in $\mathbb{R}^2$ to $30,000$ vertices in $\mathbb{R}^8$. The CVAE was implemented in TensorFlow [42] with 2 dense layers of 512 units each. Input to the CVAE is a vector encoding source and target locations and an occupancy grid. Training time over 4000 examples ranged from 20 minutes in $\mathbb{R}^2$ to 60 minutes in $\mathbb{R}^8$ problems. At test time, we time-out samplers after 5 sec. The code is open sourced[4] with more details in [39].

A. Performance Analysis

a) Sampling time: Table I reports the average time each sampler takes for 200 samples across 100 test instances. SHORTESTPATH and LEGO are the fastest. MAPRM and RBB both rely on heavy computation with multiple collision checking steps. WIS, by tetrahedralizing the workspace and identifying narrow passages, is relatively faster but slower than the learners. Unfortunately, some of the baselines time-out on manipulator planning problem due to expense of collision checking.

b) Success Rate: Table II reports the success rates (95% confidence intervals) on 100 test instances when sampling 500 vertices. Success rate is the fraction of problems for which the search found a feasible solution. LEGO has the highest success rate. The baselines are competitive in $\mathbb{R}^2$, but suffer for higher dimensional problems.

[4]https://github.com/vernwalrahul/LearningRoadmaps

c) Normalized Path Cost: This is the ratio of cost of the computed solution w.r.t. the cost of the solution on the dense graph. Fig. 6 shows the normalized cost for HALTON, SHORTESTPATH and LEGO- these were the only baselines that consistently had bounded 95% confidence intervals (i.e. when success rate is $\geq 60\%$). SHORTESTPATH has the lowest cost, however LEGO is within 10% bound of the optimal.

B. Observations

We report on some key observations from Table II and Fig. 6.

O 1. LEGO *consistently outperforms all baselines*

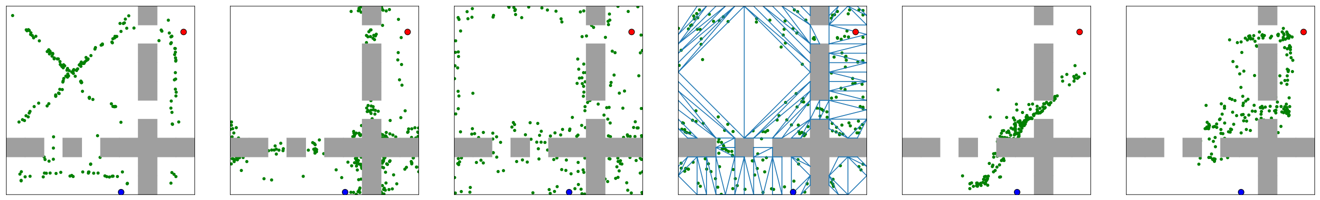
As shown in Table II, LEGO has the best success rate (for 500 samples) on all datasets. The second row in Fig. 6 shows that LEGO is within 10% bound of the optimal path.

O 2. LEGO *places samples only in regions where the optimal path may pass.*

Fig. 5 shows samples generated by various baseline algorithms on a 2D problem. The heuristic baselines use various strategies to identify important regions - MAPRM finds medial axes, RBB finds bridge points, GAUSSIAN samples around obstacles, WIS divides up space non-uniformly and samples accordingly. However, these methods places samples everywhere irrespective of the query. SHORTESTPATH takes the query into account but fails to find the gaps. LEGO does a combination of both – it finds the right gaps.

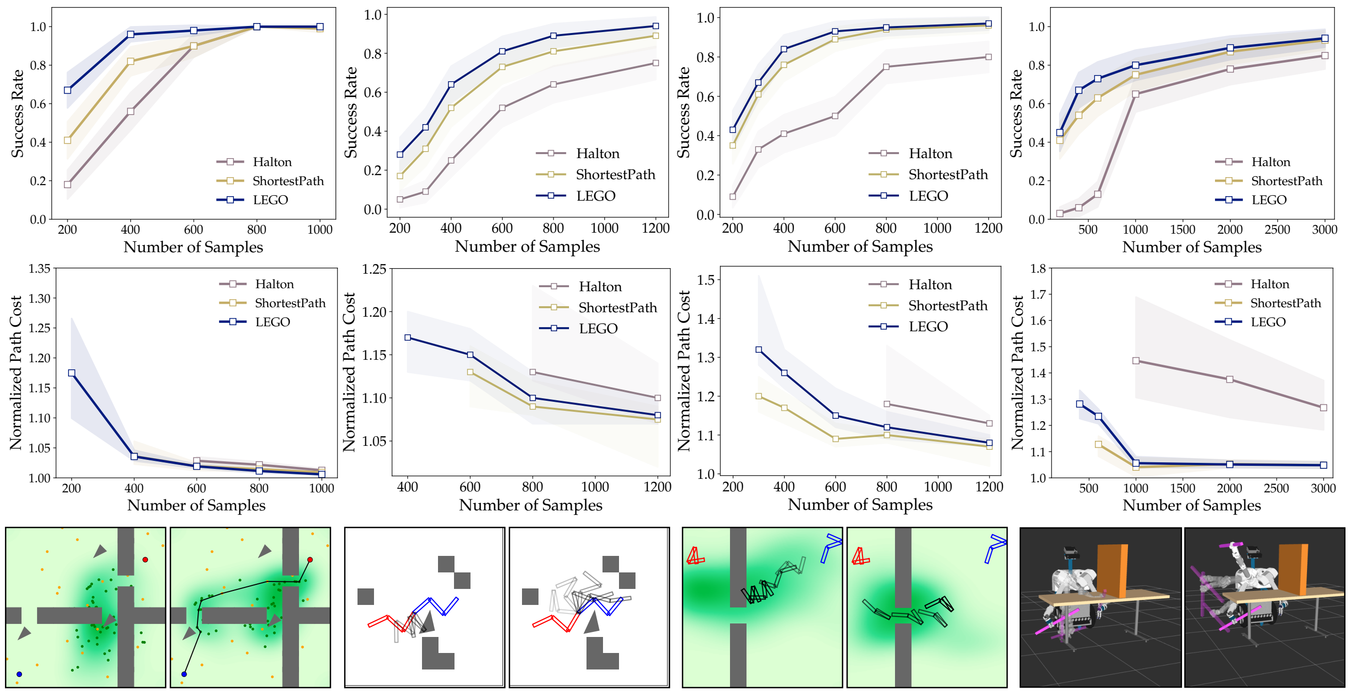
O 3. LEGO *has a higher performance gain on harder problems (narrow passages) as it focuses on bottlenecks.*

Table II shows how success rates vary in 2D problems with



(a) **MAPRM** (b) **RBB** (c) **GAUSSIAN** (d) **WIS** (e) **SHORTESTPATH** (f) **LEGO**

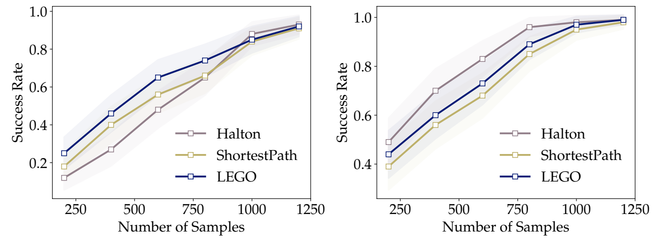
Fig. 5: Comparison of samples (green) generated by all baseline algorithms on a 2D problem, planning from start (blue) to goal (red).



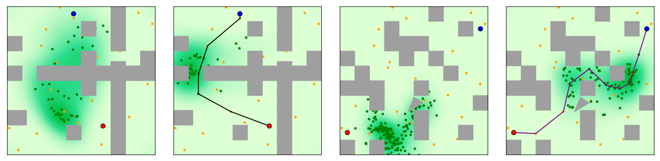
SHORTESTPATH LEGO SHORTESTPATH LEGO SHORTESTPATH LEGO SHORTESTPATH LEGO

(a) Point Robot (2D Large) (b) N-Link Arm (3D) (c) Snake Robot (5D) (d) Manipulator Constrained (8D)

Fig. 6: Comparisons of SHORTESTPATH against LEGO. Each row follows pattern (from the left): First figure shows the success rate, second figure shows the normalized path length, third figure shows SHORTESTPATH and fourth figure shows LEGO.



(a) Corrupted Environment 1 (b) Corrupted Environment 2



(c) SHORTESTPATH (d) LEGO (e) SHORTESTPATH (f) LEGO

Fig. 7: Comparison of samplers on corrupted environments, i.e., different from training dataset. Success rate on (a) less corrupted environment 1: mixture of walls and random squares and (b) more corrupted environment 2: only squares. Output of SHORTESTPATH and LEGO on environment 1 (c,d) and environment 2 (e,f).

small / medium / large gaps. As the gaps gets narrower, LEGO outperforms more dominantly. The BOTTLENECKNODE component in LEGO seeks the bottleneck regions (Fig. 4b).

For manipulator planning $\mathbb{R}^8$ problems, when stick is unconstrained, LEGO and SHORTESTPATH are almost identical. We attribute this to such problems being easier, i.e. the shortest path simply slides the stick out of the way and plans to the goal. When the stick is constrained, LEGO does far better. Fig. 6(d) shows that LEGO is able to sample around the table while SHORTESTPATH cannot find this path.

O 4. LEGO *is robust to a certain degree of train-test mismatch as it encourages diversity.*

Fig. 7 shows the success rate of learners on a 2D test environment that has been corrupted. Environment 1 is less corrupted than environment 2. Fig. 7(a) shows that on environment 1, LEGO is still the best sampler. SHORTESTPATH (Fig. 7(c)) ignores the corruption in the environment and fails. LEGO (Fig. 7(d)) still finds the correct bottleneck. Fig. 7(b) shows that all learners are worse than HALTON. SHORTESTPATH (Fig. 7(e)) densifies around a particular constrained region while LEGO (Fig. 7(e)) still finds a path due to the DIVERSEPATHSET component sampling in multiple bottleneck regions.

VII. Discussion

We present a framework for training a generative model to predict roadmaps for sampling-based motion planning. We build upon state-of-the-art methods that train the CVAE using the shortest path as target input. We identify important failure modes such as complex obstacle configurations and train-test mismatch. Our algorithm LEGO directly addresses these issues by training the CVAE using *diverse bottleneck nodes* as target input. We formally define these terms and provide provable algorithms to extract such nodes. Our results indicate that the predicted roadmaps outperform competitive baselines on a range of problems.

Using priors in planning is a double edged sword. While one can get astounding speed ups by focusing search on a tiny portion of C-space [11], any problem not covered in the dataset can lead to catastrophic failures. This is symptomatic of the fundamental problem of *over-fitting* in machine learning. While one could ensure the training data covers all possible environments [43], an algorithmic solution is to explore regularization techniques for planning. We argue DiversePathSet can be viewed as a form of regularization.

We can also include a more informed conditioning vector that captures the state of the search, e.g., the length of the current shortest path. This is similar to Informed RRT* [44]. Finally, we wish to scale to problems with varying workspace where a global planner guides the sampler to focus on relevant parts of the workspace [13, 45].

References

[1] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE TRO*, 1996.

[2] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[3] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 1968.

[4] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960.

[5] Lucas Janson, Brian Ichter, and Marco Pavone. Deterministic sampling-based motion planning: Optimality, complexity, and performance. *arXiv preprint arXiv:1505.00023*, 2015.

[6] Christopher Holleman and Lydia E Kavraki. A framework for using the workspace medial axis in PRM planners. In *ICRA*, 2000.

[7] D Hsu, G Sánchez-Ante, and Z Sun. Hybrid prm sampling with a cost-sensitive adaptive strategy. In *ICRA*, 2005.

[8] Brendan Burns and Oliver Brock. Sampling-based motion planning using predictive models. In *ICRA*, 2005.

[9] D Hsu, J Latombe, and H Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *IJRR*, 2006.

[10] H Kurniawati and D Hsu. Workspace-based connectivity oracle: An adaptive sampling strategy for prm planning. In *Algorithmic Foundation of Robotics VII*, pages 35–51. Springer, 2008.

[11] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *ICRA*, 2018.

[12] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[13] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *International Journal Computational Geometry & Applications*, 4:495–512, 1999.

[14] Steven A Wilmarth, Nancy M Amato, and Peter F Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *ICRA*, 1999.

[15] D Hsu, T Jiang, J Reif, and Z Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *ICRA*, 2003.

[16] H Kurniawati and D Hsu. Workspace importance sampling for probabilistic roadmap planning. In *IROS*.

[17] Yuandong Yang and Oliver Brock. Adapting the sampling distribution in prm planners based on an approximated medial axis. In *ICRA*, 2004.

[18] Jur P Van den Berg and Mark H Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. *IJRR*, 2005.

[19] Nancy M Amato, O Burchan Bayazit, and Lucia K Dale. OBPRM: An obstacle-based PRM for 3D workspaces. 1998.

[20] Valérie Boor, Mark H Overmars, and A Frank Van Der Stappen. The gaussian sampling strategy for probabilistic roadmap planners. In *ICRA*, pages 1018–1023, 1999.

[21] M Morales, L Tapia, R Pearce, S Rodriguez, and N M Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*, pages 361–376. Springer, 2005.

[22] S Rodriguez, S Thomas, R Pearce, and N M Amato. Resampl: A region-sensitive adaptive motion planner. In *Algorithmic Foundation of Robotics VII*, pages 285–300. Springer, 2008.

[23] Sébastien Dalibard and Jean-Paul Laumond. Linear dimensionality reduction in random motion planning. *IJRR*, 2011.

[24] Jia Pan, Sachin Chitta, and Dinesh Manocha. Faster sample-based motion planning using instance-based learning. In *WAFR*, 2012.

[25] Shushman Choudhury, Christopher M Dellin, and Siddhartha S Srinivasa. Pareto-optimal search over configuration space beliefs for anytime motion planning. In *IROS*, 2016.

[26] Xinyu Tang, Jyh-Ming Lien, and Nancy Amato. An obstacle-based rapidly-exploring random tree. In *ICRA*, 2006.

[27] Brendan Burns and Oliver Brock. Toward optimal configuration space sampling. In *RSS*, 2005.

[28] Rosen Diankov and James Kuffner. Randomized statistical path planning. In *IROS*, 2007.

[29] Matt Zucker, James Kuffner, and J Andrew Bagnell. Adaptive workspace biasing for sampling-based planners. In *ICRA*, 2008.

[30] Yen-Ling Kuo, Andrei Barbu, and Boris Katz. Deep sequential models for sampling-based planning. *arXiv preprint arXiv:1810.00804*, 2018.

[31] Paul Vernaza and Daniel D Lee. Learning dimensional descent for optimal motion planning in high-dimensional spaces. In *AAAI*, 2011.

[32] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[33] N Chen, M Karl, and P van der Smagt. Dynamic movement primitives in latent space of time-dependent variational autoencoders. In *Humanoids*, 2016.

[34] Jung-Su Ha, Hyeok-Joo Chae, and Han-Lim Choi. Approximate inference-based motion planning by learning and exploiting low-dimensional latent variable models. *IEEE RAL*, 2018.

[35] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *arXiv preprint arXiv:1807.10366*, 2018.

[36] C Zhang, J Huh, and D D Lee. Learning implicit sampling distributions for motion planning. *arXiv preprint arXiv:1806.01968*, 2018.

[37] A H Qureshi and M C Yip. Deeply informed neural sampling for robot motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6582–6588. IEEE, 2018.

[38] K Sohn, H Lee, and X Yan. Learning structured output representation using deep conditional generative models. In *NIPS*, 2015.

[39] R Vernwal, A Mandalika, S Choudhury, and S Srinivasa. Supplementary: Learning to sample efficient roadmaps. 2018.

[40] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 1998.

[41] S. Srinivasa, D. Berenson, M. Cakmak, A. C. Romea, M. Dogar, A. Dragan, R. Knepper, T. Niemueller, K. Strabala, J. M. Vandeweghe, and J. Ziegler. Herb 2.0: Lessons learned from developing a mobile manipulator for the home. *Proceedings of the IEEE*, 100(8):1–19, July 2012.

[42] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. URL https://www.tensorflow.org/.

[43] J Tobin, R Fong, A Ray, J Schneider, W Zaremba, and P Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, 2017.

[44] J D Gammell, S S Srinivasa, and T D Barfoot. Informed RRT*: Optimal incremental path planning focused through an admissible ellipsoidal heuristic. In *IROS*, 2014.

[45] Stephan Zheng, Yisong Yue, and Jennifer Hobbs. Generating long-term trajectories using deep hierarchical networks. In *NIPS*, 2016.