Reducing Collision Checking for Sampling-Based Motion Planning Using Graph Neural Networks

Anonymous Author(s)
Affiliation
Address
email

Abstract

Sampling-based motion planning is a popular approach in robotics for finding paths in continuous configuration spaces. Checking collision with obstacles is the major computational bottleneck in this process. We propose new learning-based methods for reducing collision checking to accelerate motion planning by training graph neural networks (GNNs) that perform path exploration and path smoothing. Given random geometric graphs (RGGs) generated from batch sampling, the path exploration component iteratively predicts collision-free edges to prioritize their exploration. The path smoothing component then optimizes paths obtained from the exploration stage. The methods benefit from the ability of GNNs of capturing geometric patterns from RGGs through batch sampling and generalize better to unseen environments. Experimental results show that the learned components can significantly reduce collision checking and improve overall planning efficiency in challenging high-dimensional motion planning tasks.

1 Introduction

Sampling-based planning is a popular approach to high-dimensional continuous motion planning in robotics [32, 11, 27, 23, 16, 47, 31]. The idea is to iteratively sample configurations of the robots and construct one or multiple exploration trees to probe the free space, such that the start and goal states are eventually connected by some collision-free path through the sampled states, ideally with path cost minimized. This motion planning problem is hard, theoretically PSPACE-complete [42], and existing algorithms are challenged when planning motions of robots with a few degrees of freedom [30, 12, 1, 35, 7, 12]. In particular, the planning algorithms need to repeatedly check whether an edge connecting two sample states is feasible, i.e., that no state along the edge collides with any obstacle. This *collision checking* operation is the major computational bottleneck in the planning process and by itself NP-hard in general [24, 5]. For instance, consider the 7D Kuka arm planning problem in the environment shown in Figure 1. The leading sampling-based planning algorithm BIT* [16] spends about 28.6 seconds to find a complete motion plan for the robot, in which 20.2s (70.6% of time) is spent on collision checking. In comparison, it only takes 0.06s (0.2% of time) for sampling all the probing states needed for constructing random graphs for completing the search.

Learning-based approaches have become popular for accelerating motion planning. Many recent approaches learn patterns of the configuration spaces to improve the sampling of the probing states, typically through reinforcement learning or imitation learning [25, 21, 58, 41, 6]. For instance, Ichter et al. [21] and motion planning networks [41] apply imitation learning on collected demonstrations to bias the sampling process. The NEXT algorithm [6] provides a state-of-the-art design for embedding high-dimensional continuous state spaces into low-dimensional representations, while balancing exploration and exploitation in the sampling process. It has demonstrated clear benefits of using learning-based components to reduce samples and accelerate planning. However, we believe two

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

aspects in NEXT can be improved, if we shift the focus from reducing *sample complexity* to reducing *collision checking*. First, instead of taking the grid-based encoding of the entire workspace as input, we can use the graphs formed by batches of samples from the free space to better capture the geometric patterns of an environment. Second, having access to the entire graph formed by samples allows us to better prioritize edge exploration and collision checking based on relatively global patterns, and avoid getting stuck in local regions. In short, with reasonably relaxed budget of samples taken uniformly from the space, we can better exploit global patterns to reduce the more expensive collision checking operations instead. Figure 1 shows a typical example of how the trade-off benefits overall planning, and more thorough comparisons are provided in the experimental results section.

We design two novel learning-based components that utilize Graph Neural Networks (GNNs) to accelerate the search for collision-free paths in batch sampling-based motion planning algorithms. The first component is the GNN Path Explorer, which is trained to find collision-free paths given the environment configuration and a random geometric graph (RGG) formed by probing samples. The second component is the GNN Path Smoother, which learns to optimize the path obtained from the explorer. In both models, we rely on the expressiveness and permutation invariance of GNNs as well as attention mechanisms to identify geometric patterns in the RGGs formed by samples, and accelerate combinatorial search.

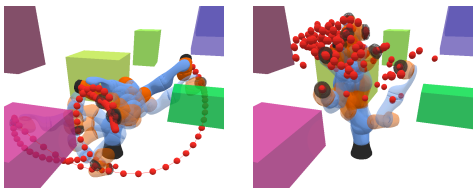

Figure 1: Performance on 7D Kuka arm. Left: Trajectory generated by the proposed GNN-based approach. Right: NEXT getting stuck at a local region. Both methods were trained on the same training set.

The proposed learning-based components can accelerate batch sampling-based planning algorithms without compromising probabilistic completeness properties. The methods achieve higher success rate, much lower rate of collision checking, and accelerate the overall planning compared to state-of-the-art methods. We evaluate the proposed approach in a variety of planning environments from 2D mazes to 14D dual KUKA arms. Experiments show that our method significantly reduces collision checking, improves the overall planning efficiency, and scales well to high-dimensional tasks.

The paper is organized as follows. We review related work and preliminaries in Section 2 and 3. We describe the detailed design of the GNN architectures in Section 4, followed by the training of GNN explorer and smoother in Section 5. We discuss experimental results in Section 6.

2 Related Work

Learning-based Motion Planning. Learning-based approaches typically consider motion planning as a sequential decision-making problem, which could be naturally incorporated with reinforcement learning or imitation learning. With model-based reinforcement learning, DDPG-MP [25] integrates the known dynamic of robots and trains a policy network. Strudel et al. [48] improves the obstacle encoding with the position and normal vectors. OracleNet [2] learns via oracle imitation and encodes the trajectory history by an LSTM [19]. Other than predicting nodes which are expected to greedily form a trajectory, various approaches have been designed to first learn to sample vertices, then utilize sampling-based motion planning for further path exploration through these samples. L2RRT [20] first embeds high-dimensional configuration into low-dimensional representation, then performs RRT [32] on top of that. Ichter et al. [21] uses conditional VAE to sample nodes. Zhang et al. [58] learns a rejection sampling distribution. Madaan [36] encodes the explored tree with an RNN. Motion Planning Networks [41] utilizes the dropout-based stochastic auto-encoder for biased sampling. NEXT [6] projects the high-dimensional planning spaces into low-dimensional embedding discrete spaces, and further applies Gated Path Planning Networks [33] to predict the samples.

Existing learning-based approaches have considered improving *collision detection*. Fastron [10] and ClearanceNet [4] learn function approximators as a proxy to collision detection, which is disparate from our focus on reducing the steps that are needed to the collision checker, and can be improved further potentially if combined together. Another recent line of work focuses on learning to explore edges given fixed graph. Value Iteration Networks [49] and Gated Path Planning Networks [33] apply convolutional neural networks (CNN) on discrete maps, then predict the policy with a weighted attention sum over neighborhoods. Generalized Value Iteration Networks [38] and Velickovic et al.

[53] extend this approach for nontrivial graph by replacing CNN with GNN. However, the construction of such graphs requires ground-truth collision status for every edge on the graph at inference time.

It should be noted that other than sampling-based approaches, trajectory optimization [26, 61, 51, 45, 60], and motion primitives [37, 59] are standard choices for more structured problems such as for autonomous cars and UAVs, while sampling-based methods are important for navigating high-dimensional cluttered spaces such as for manipulators and rescue robots.

Graph Neural Networks for Motion Planning. Graph neural networks are permutation invariant to the orders of nodes on graph, which become a natural choice for learning patterns in graph problems. They have been successfully applied in robotics applications such as decentralized control [34]. For sampling-based motion planning, Khan et al. [29] utilizes GNN to identify critical samples. We focus on the different aspect of collision checking with given random geometric graphs, and can be combined with existing techniques without affecting probabilistic completeness. More broadly, GNNs have been used for learning patterns in general graph-structured problems, e.g. graph-based exploration [9, 46], combinatorial optimization [28, 13, 3], neural algorithm execution [53, 54, 56, 52]. Other than to use GNN for high-dimensional planning, several works propose to first learn neural metrics, then build explicit graphs upon the learned metric which is used later to search the path [44, 15, 14, 57]. While sharing similar interests, our work specifically focus on how to reduce the collision checking steps for sampling-based motion planning.

Informed Sampling for Motion Planning. A main focus in motion planning is on developing problem-independent heuristic functions for prioritizing the samples or edges to explore. Approaches include Randomized A* [11], Fast Marching Trees (FMT*) [23], Sampling-based A*(SBA*) [39], Batch Informed Trees (BIT*) [16]. These methods are orthogonal to our learning-based approach, which can further exploit the problem distribution and recognize patterns through offline training to improve efficiency. Recent work in motion planning has made significant progress in reducing collision checking through batch sampling and incremental search, such as in BIT* [16] and AIT *[47]. The idea is to start with batches of probing random samples in the free space, and focus on reducing collision checking to edges that are likely on good paths to the goal, which also inspires our work.

3 Preliminaries

Motion Planning. We focus on motion planning in continuous spaces, where the *configuration space* is $C \subseteq \mathbb{R}^n$. The configuration space includes all degrees of freedom of a robot (e.g. all joints of a robotic arm) and is different from the *workspace* where the robot physically resides in, which is at most 3-dimensional. For planning problem on a graph $G = \langle V, E \rangle$, we denote the start vertex and goal vertex as $v_s, v_g \in C$. A path from v_s to v_g is a finite set of edges $\pi = \{e_i : (v_i, v_i')\}_{i \in [0,k]}$ such that $v_0 = v_s$, $v_k' = v_g$, and $v_i' = v_{i+1}$ for all $i \in [0, k-1]$. An environment for a motion planning problem consists of a set of obstacles $C_{obs} \subseteq C$ and free space $C_{free} = C \setminus C_{obs}$. Note that C_{obs} is the projection of 3D objects in the higher-dimensional configuration space, and typically has complex geometric structures that can not be efficiently represented. A sample state $v \in C$ in the configuration space is free if $v \in C_{free}$, i.e., it is not contained in any obstacle. An edge connecting two samples is free if $e \subseteq C_{free}$. Namely, for every point v on the edge e, $v \in C_{free}$. A path π is free if all its edges are free. A random geometric graph (RGG) is a r-disc or k-nearest-neighbor (k-NN) graph G [17, 55], where nodes are randomly sampled from the free space C_{free}. In this paper we consider the RGG as a k-NN graph. Given a random geometric graph G and a pair of start and goal configuration (v_s, v_g), the goal of agent is to find a free path π from v_s to v_g. Without loss of generality, we consider the cost of a path to be the total length over all edges in it.

Graph Neural Networks and Attention. Let $G = \langle V, E \rangle$ be a finite graph where each vertex v_i is labeled by data $x_i \in \mathbb{R}^n$. A graph neural network (GNN) learns the representation h_i of node v_i by aggregating the information from its neighbors $\mathcal{N}(v_i)$. Given fully-connected networks f and g, a typical GNN encodes the representation $h_i^{(k+1)}$ of the node v_i after k aggregation as:

$$c_i^{(k)} = \oplus^{(k)}(\Big\{ f(h_i^{(k)}, h_j^{(k)}) \mid (v_i, v_j) \in E \Big\}) \text{ and } h_i^{(k+1)} = g(h_i^{(k)}, c_i^{(k)}) \tag{1}$$

where $h_i^{(1)} = x_i$ and $\oplus$ is some permutation-invariant aggregation function on sets, such as max, mean, or sum. We will also use the attention mechanism when we need to encode a varied number of

obstacles as inputs. Suppose there are n keys each with dimension d_q: $K \in \mathbb{R}^{n \times d_k}$, each key corresponding to a value $V \in \mathbb{R}^{n \times d_v}$. Given m query vectors $Q \in \mathbb{R}^{m \times d_k}$, a typical attention function $\text{Att}(K, Q, V)$ returns the representation for each query as $\text{Att}(K, Q, V) = \text{softmax}(QK^T/\sqrt{d_k})V$. The function is also permutation-invariant so the order of obstacles does not affect the output.

4 GNN Architecture for Path Exploration and Smoothing

4.1 Overall Approach

At a high level, motion planning with batch sampling typically proceeds as follows [16]. We first sample a batch of configurations in the free space, together with the start and goal states, and form a random graph (RGG) by connecting neighbor states (such as k-NN). A tree is then built in this graph from the start towards the goal via heuristic search. The tree can only contain collision-free edges, so each connection requires collision checking. When a path from start to goal is found, it is stored as a feasible plan, which can be later updated to minimize cost. After adding all collision-free edges in the current batch in the tree, a new batch will be added and the tree is further expanded. The algorithm keeps sampling batches and expanding the tree until the computation budget is reached. It then returns the best path found so far, or failure if none has been found.

We use GNN models to improve two important steps in this planning procedure: path exploration and path smoothing. The GNN path explorer finds a feasible path π from start to goal given a randomly batch-sampled RGG, with the goal of reducing the number of edges that need to be checked for collision. The GNN path smoother then takes the path found by the explorer and attempts to produce another feasible path with less cost. In both tasks, the GNN models aim to recognize patterns and learn solutions to the combinatorial problems and save computation. In Figure 2, we illustrate the main steps for the overall algorithm. First, in (a-c), we generate an RGG composed of the vertices randomly sampled from the free space *without collision checking* on edges. This graph will provide patterns that the GNN explorer later uses to only prioritize certain edges for collision checking. In (d), the graph is the input to the GNN path explorer, which predicts the priority of the edges to explore and only the proposed edges are checked for collision. (e): We iteratively query the path explorer with collision checking to expand a tree in the free space until the goal vertex is reached, and sample new batches when no path is found in the current graph. (f): Once a path is provided by the path explorer, it becomes the input (together with the current RGG) to the GNN path smoother component, which outputs a new path that reduces path cost via local changes to the vertices in the input path.

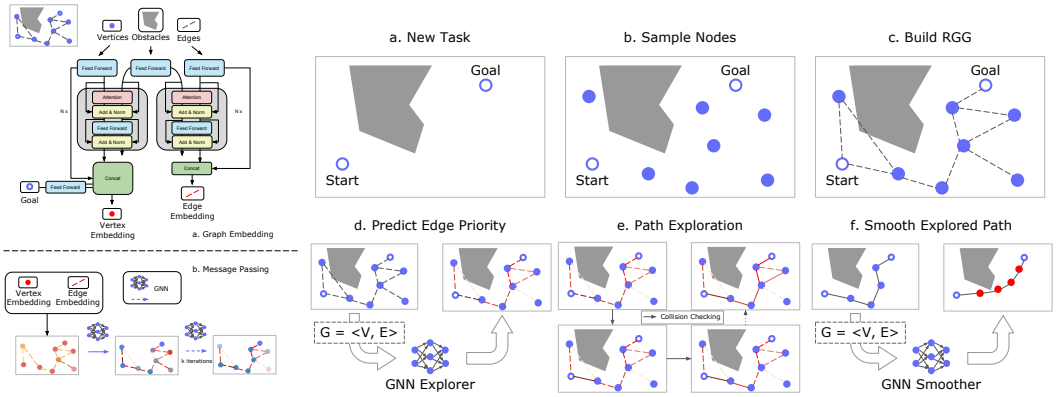

Figure 2: Left: GNN architecture shared by the path explorer and the path smoother. Right (a-f): Main steps in planning with GNNs, as explained in Section 4.1.

4.2 GNN Architecture

We write the GNN path explorer as $\mathcal{N}_E$ and the GNN path smoother as $\mathcal{N}_S$. Both models take in a sampled random geometric graph $G = \langle V, E \rangle$. For an n-dimensional configuration space, each vertex $v_i \in \mathbb{R}^{n+3}$ contains an n-dimensional configuration component and a 3-dimensional one-hot label. There are 3 kinds of labels for the explorer: (i) the vertices in the free space, (ii) the vertices

with collision, and (iii) the special goal vertex. There are also 3 kinds of labels for the smoother: (i) the vertices on the path, (ii) the vertices in the free space, and (iii) the vertices with collision.

The vertices and the edges are first embedded into a latent space with $x \in \mathbb{R}^{|V| \times d_h}, y \in \mathbb{R}^{|E| \times d_h}$, where d_h is the size of the embedding. The embeddings for the GNN explorer and smoother are different, which will be discussed later in this section. Taking the vertex and edge embedding x, y, the GNN aggregates the local information for each vertex from the neighbors, by performing the following operation with 2 two-layer MLPs f_x and f_y:

$$\begin{aligned} x_i &= g\left(x_i, \max\{f_x(x_j - x_i, x_j, x_i, y_l) \mid e_l : (v_i, v_j) \in E\}\right), \forall v_i \in V \\ y_l &= \max(y_l, f_y(x_j - x_i, x_j, x_i)), \forall e_l : (v_i, v_j) \in E \end{aligned} \tag{2}$$

Note that here we use $\max$ as the aggregation operator to gather the local geometric information, due to its empirical robustness to achieve the order invariance [40]. The edge information is also incorporated by adding y_l as the input to f_x. The update function g is implemented in two different ways for the GNN explorer and smoother. Specifically, g equals to the $\max$ operator for the GNN explorer, and $g(m_i, x_i) = f_g(m_i) + x_i$ as the residual connection for the GNN smoother, where f_g is a two-layer MLP. We choose $\max$ operator for the explorer, due to its inductive bias to imitate the value iteration, as mentioned by Velickovic et al. [53]. The residual connection is applied to the smoother, since intuitively the residual provides a direction for the improvement of each node on the path in the latent space, which fits our purpose to generate a shorter path for the smoother.

We also note that Equation 2 directly updates on the x and y and is a homogeneous function similar to Tang et al. [50], which allows us to self-iterate x and y over multiple loops without introducing redundant layers. Both the GNN explorer and smoother leverage this property. After several iterations, with two MLPs $f_\eta, f_u, \mathcal{N}_E$ outputs the priority $\eta = f_\eta(y)$ for each edge, and $\mathcal{N}_S$ outputs a potentially shorter path $\pi' = \{u_i, u'_i\}, u_i = f_u(x_i)$ for $v_i \in \pi$.

Special design for the GNN path explorer. The path explorer uses the embedding of the vertices of the form $x = h_x(v, v_g, (v - v_g)^2, v - v_g)$, where h_x is a two-layer MLP with batch normalization [22]. Here we append the L2 distance and the difference to the goal to the vertex embedding, which serve as heuristics for the GNN to be more informed about which node is more valuable. The y_l is simply computed as $y_l = h_y(v_j - v_i, v_j, v_i)$, where h_y is also a two-layer MLP with batch normalization. Optionally, it is helpful for the explorer to incorporate the configuration of obstacles $O = \{\mathbf{o}\} \in \mathbb{R}^{|\{\mathbf{o}\}| \times 2n}$ as inputs, when embedding the vertices and edges. Since the obstacles of the environment has variable numbers, we utilize the attention mechanism here to update the x and y, named as *obstacle encoding*, as illustrated in Figure 2. Further details are provided in the Appendix.

Special design for GNN path smoother. The GNN smoother embeds vertices with $x = h_x(v)$, where h_x is a two-layer MLP with batch normalization. The y_l is computed as $y_l = h_y(v_j - v_i, v_j, v_i)$, where h_y is a two-layer MLP with batch normalization. Each time x and y are updated by Equation 2, the GNN smoother will output a new smoother path $\pi' = \{(u_i, u'_i)\}_{i \in [0,k]}$, where $u_i = f_u(x_i), \forall v_i \in \pi$, given an MLP f_u. The u_0 and u'_k are manually replaced by v_s and v_g, to satisfy the path constraint. We assume the π' has the same number of nodes as π. Since the GNN smoother could gain novel local geometric information with the changed vertices of the new path, we dynamically update $G = \langle V, E \rangle$, via (i) replacing those nodes labeled as path nodes in V by the nodes on new path, (ii) replacing E by generating a k-NN graph on the updated V. With the updated graph G, we repeat the above operation. During training, the GNN smoother outputs π', after a random number of iterations (between 1 and 10). During evaluation, the GNN smoother outputs π' after only one loop for each calling.

5 Training the Path Explorer and Smoother

Due to space limitation we provide the pseudocode for all algorithms in the Appendix.

5.1 GNN Explorer $\mathcal{N}_E$: Training and Inference

The path explorer constructs a tree through sampled states with the hope of reaching the goal state in a finite number of steps. We initialize the tree $\mathcal{T}_0$ with the start state v_s as its root. Every edge $e_{\mathcal{T}_i}$ in the tree $\mathcal{T}_i$ exists only if $e_{\mathcal{T}_i}$ is in the free space C_{free}. Given an RGG $G = \langle V, E \rangle$, Our goal is to find a tree containing the goal configuration v_g by adding edges from E to the tree, with as few

collision checks as possible. We write the edge on frontier of the tree as $E_f(\mathcal{T}) = \{(v_i, v_i') \mid v_i \in V_\mathcal{T}, v_i' \notin V_\mathcal{T}\}$. We denote the set of edges with unknown collision status at time step i as E_i.

Training procedures. Each training problem consists of a set of obstacles O, start vertex v_s, goal vertex v_g, we sample a k-NN graph $G = \langle V, E \rangle$, where V is the random vertices sampled from the free space combined with $\{v_s, v_g\}$. The goal is to train $\mathcal{N}_E$ to predict exploration priority $\eta \in \mathbb{R}^{|E|}$.

A straightforward way for supervision is to use the Dijkstra's algorithm to compute the shortest feasible path from v_s to v_g, and maximize the corresponding values of η at the edges of this path, via cross entropy loss or Bayesian ranking [43]. However, it does not provide useful guidance when the search tree deviates from the ideal optimal path at inference time. Instead, we first explore the graph using η with i steps, which forms a tree $\mathcal{T}_i$, where i is a random number. The oracle provides the shortest feasible path π_N in this tree and connects one of the nodes on $\mathcal{T}_i$ to the goal vertex v_g. We formulate this optimal path as $\pi_N = \{e_{N_i} : (v_{N_i}, v'_{N_i})\}_{i \in [0,k]}$, where $v_{N_0} \in V_{\mathcal{T}_i}, v'_{N_k} = v_g$. We train the explorer to imitate this oracle. Namely, the explorer will directly choose $e_{N_0} \in \pi_N$ as the next edge to explore, among all possible edges on the frontier of $\mathcal{T}_i$, i.e. $E_i \cap E_f(\mathcal{T}_i)$. We maximize the η_{N_0} among the values of $\{\eta_i \mid e_i \in E_i \cap E_f(\mathcal{T}_i)\}$ using the following cross entropy loss:

$$L_{\mathcal{N}_E} = -\log \gamma_{N_0}, \text{ where } \gamma_k = \frac{e^{\eta_k}}{\sum_{e_j \in E_i \cap E_f(\mathcal{T}_i)} e^{\eta_j}}, \forall e_k \in E_i \cap E_f(\mathcal{T}_i) \tag{3}$$

Inference procedures. Given the GNN $\mathcal{N}_E$, the current explored tree $\mathcal{T}_i$ at step i, the RGG $G = \langle V, E \rangle$ including v_s and v_g, environment configuration O, GNN path explorer aims to maximize the probability of generating a feasible path by adding e_i from E_i to tree $\mathcal{T}_i$ as:

$$e_i = \underset{e_k \in E_i \cap E_f(\mathcal{T}_i)}{\arg\max} \; \mathcal{N}_E(\eta_k \mid V, E, O) \tag{4}$$

where η_k is the output of $\mathcal{N}_E$ for the edge e_k. After e_i is proposed by GNN using Equation 4, we check the collision of e_i. If e_i is not in collision with the obstacles, we add the edge e_i to the tree $\mathcal{T}_i$, and remove e_i from E_i, i.e., $E_{\mathcal{T}_{i+1}} = E_{\mathcal{T}_i} \cup \{e_i\}$, and $E_{i+1} = E_i \setminus \{e_i\}$. If e_i is in collision with obstacles, we query the path explorer for the next proposed edge using Equation 4, where E_i is updated as $E_i = E_i \setminus \{e_i\}$. The loop terminates when we find a collision-free edge, or when $E_i \cap E_f(\mathcal{T}_i) = \emptyset$. When the latter happens, we re-sample another batch of samples, add new samples to vertices V, re-construct k-NN graph G, re-compute η, and continue to explore the path on this new graph with the explored nodes and edges.

The exploration GNN only proposes an ordering on the candidate edges, and all possible edges may still be collision checked in the worst case. Thus, if there exists any complete path in the RGG, the algorithm always finds it. Therefore, the proposed learning-based component does not affect the probabilistic completeness of sampling-based planning algorithms [8].

5.2 GNN Path Smoother $\mathcal{N}_S$: Training and Inference

The GNN $\mathcal{N}_S$ for path smoothing takes an RGG and a path π proposed by the explorer, and aims to produce a shorter path π'. Specifically, the input is a graph $G = \langle V, E \rangle$, where $V = V_\pi \cup V_f \cup V_c$, $E = E_\pi \cup E_{fc}$. Here, V_f and V_c are reused as the same vertices in the GNN explorer, without introducing extra sampling complexity. E_π is composed of those pairs of the adjacent vertices on π, and E_{fc} connects each vertex in V_π to their k-nearest neighbor in $V_f \cup V_c$. Intuitively, aggregating information from $V_f \cup V_c$ can allow GNN to identify local regions that provide promising improvement on the current path, and avoid those that may yield potential collision.

Training procedures. We train the GNN path smoother $\mathcal{N}_S$ by imitating a smoothing oracle $\mathcal{S}$ similar to the approach of gradient-informed path smoothing proposed by Heiden et al. [18]. To prepare the training set, we iteratively perform the following two operations on each training sample path. Given a feasible path π predicted by $\mathcal{N}_E$, the smoothing oracle first tries to move the nodes on the path π with perturbation within range ϵ. If the new path π_M is feasible and has cost less than π, then $\mathcal{S}$ will continue to smooth on π_M. Otherwise, $\mathcal{S}$ will continue smoothing on π via random perturbation. After several perturbation trials, the oracle further attempts to connect pairs of nonadjacent nodes directly by a line segment. If such a segment is free of collision, then the original intermediate nodes will be moved on this linear segment. Further details are in the Appendix.

After $\mathcal{S}$ reaches maximum iteration, the oracle will return the smoothed path $\pi_M = \{w_i, w'_i\}_{i\in[0,k]}$. To generate training data, we first run our trained GNN explorer for each training problem to get the initial path π. Then the oracle path π_M and the predicted path π' are computed, and finally the GNN is trained via minimizing the MSE loss $L_{\mathcal{N}_S} = \frac{1}{k}\sum_{i\in[1,k]} \|\mathcal{N}_S(u_i \mid V, E) - w_i\|_2^2$.

Inference procedures. The GNN $\mathcal{N}_S$ for path smoothing takes an explored path $\pi : \{(v_i, v'_i)\}_{i\in[0,k]}$, a sampled graph G, and outputs a potentially shorter path $\pi' : \{(u_i, u'_i)\}_{i\in[0,k]}$ which has the same number of edges as π. It is not always guaranteed that π' is collision-free. However, such π' still indicates directions for shortening the path, so we can improve π towards π' in an incremental way. The GNN smoother will try to move every node v_i towards the target position u_i, with a small step size ϵ. For each time that all the vertices are moved with a small step, we check whether each edge still holds collision-free. If not, then the vertices on the edge will undo the movement. Otherwise, the new configuration of the vertex will replace its old configuration on π. This operation will be iterated over several times, until the maximum iteration is reached, or no edges on π can be moved further. We can then repeat the process by feeding the updated π back to the GNN $\mathcal{N}_S$ for further improvement. The intuition here is that there might still be chances to improve upon the updated π, by aggregating new information from its changed neighborhoods. This has shown empirical advantage in our experiments.

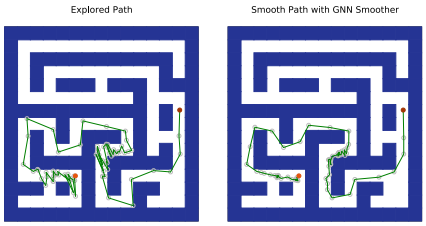

Figure 3: GNN path smoother on the 2D maze problem. It learns to improve the explored path and achieves lower cost.

6 Experiments

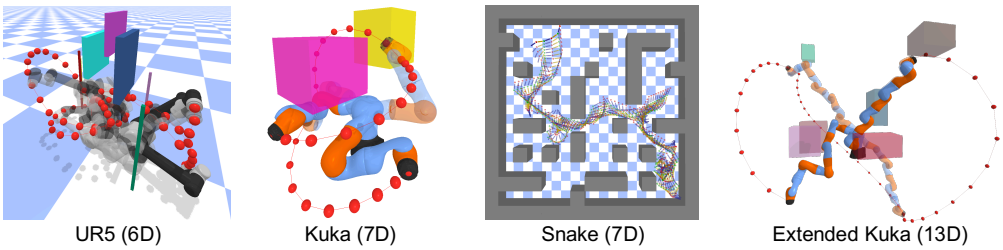

Figure 4: Demonstrations for some of our environments from UR5 to 13D.

6.1 Overall Performance

We compare our methods with the sampling-based planning baseline RRT* [27], the state-of-the-art batch-sampling based method BIT* [16], and the state-of-the-art learning-based method NEXT [6]. NEXT has been shown in [6] to outperform competing learning-based approaches. We conduct the experiments on the following environments: (i) a 2D point-robot in 2D workspace, (ii) a 6D UR5 robot in a 3D workspace, (iii) a 7D snake robot in 2D workspace (the z-axis is fixed), (iv) a 7D KUKA arm in a 3D workspace, (v) an extended 13D KUKA arm in 3D workspace, (vi) and a pair of 7DoF KUKA arms (14 DoF) in 3D workspace.

For each environment, we randomly generate 2000 problems for training and 1000 problems for testing. Each problem contains a different set of random obstacles, and a pair of feasible v_s and v_g. We run all experiments over 4 random seeds. The averaged results are illustrated in Figure 5. For the 2D environment, we directly take the training set provided by NEXT to train our GNN. We use two test sets for the 2D environment: “Easy2D" is the original test set used for evaluating NEXT in the original paper [6], and “Hard2D" is a new set of tests we generated by requiring the distance between the start and goal to be longer than the easy environments. The goal is to test whether the learned models can generalize to harder problems without changing the training set. Further details on the environments and hyperparameters are provided in the Appendix.

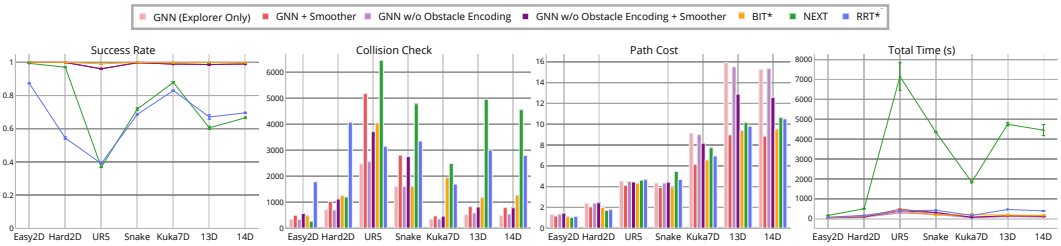

Figure 5: Comparison of performances on all environments from 2D to 14D, averaged over 4 random seeds. From left to right: (a) Success rate. (b) Collision checks. (c) Path cost. (d) Total planning time.

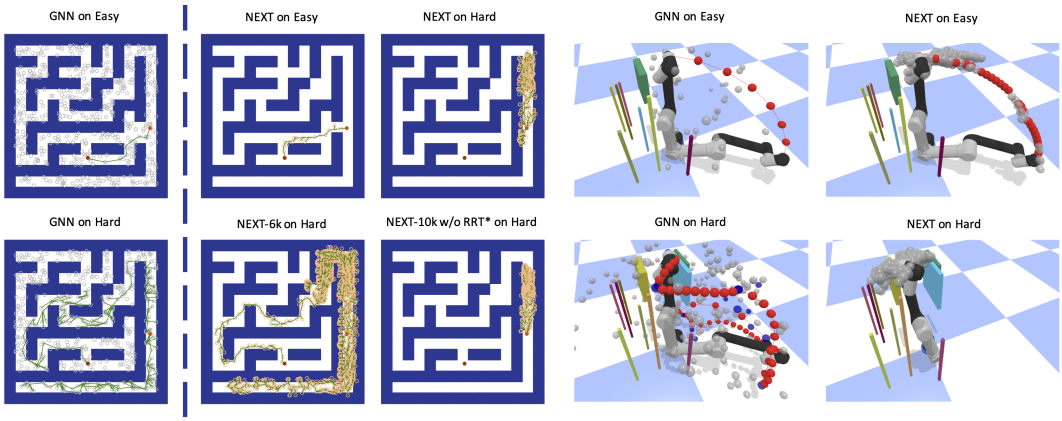

Figure 6: We test the generalization capability of GNN-based approaches and NEXT by constructing pairs of problems that have small but important difference in connectivity. The GNN models find paths on both environments quickly, while NEXT gets stuck in hard instances because of the lack of access to the graph structure provided by probing samples. The explored vertices of GNN on UR5 environment are colored in blue, and the edges on the path are colored in red.

Success rate. As shown in Figure 5 (a), our method finds complete paths at 100.0% problems on both 2D Easy and 2D Hard, and at 97.18%, 99.85%, 99.20%, 99.15%, and 99.15% problems from UR5 to 14D, which is comparable to handcrafted heuristics used in BIT* (100.0%, 100.0%, 99.25%, 99.85%, 99.65%, 99.92%, 99.82% on each environment). The learning-based planner NEXT performs well on easy 2D problems (99.37% success rate), but drops slightly to 97.10% on harder 2D problems, and 36.80%, 71.80%, 87.9%, 60.52%, 66.57% on environments in higher dimensions.

Collision checking. In Figure 5 (b), we see significant reduction of collision checking using the proposed approaches, in comparison to other approaches especially in high-dimensional problems. The average number of collision checks by the GNN explorer is 336.3, 715.7, 2474.0, 1602.2, 350.5, 521.7 and 487.0 on the given sets of environments, whereas BIT* needs 112%, 175%, 164%, 101%, 557%, 226%, 263% times as many collision checks as our method requires on each environments. NEXT requires 270.23 checks on Easy2D and increases to 1206.1 on Hard2D. On the 14D environment, our method uses 17.4% of the collision checks of what is needed by NEXT.

Path cost. We show the average path cost over all problems where all algorithms successfully found complete paths. With the smoother and obstacle encoding, our GNN approach provides the best results from UR5 to 14D, and generates comparable results for the 2D Easy and 2D Hard, where NEXT is 1.02, 1.71, and GNN is 1.18, 2.05. We find that although the GNN explorer does not yield shorter path with obstacle encoding, these explored paths can be improved further with the smoother. The reason may be that with additional obstacle encoding, the GNN explorer tends to explore edges with less probability of collision and provides more space for the smoother to improve the paths.

Planning time. A common concern about learning-based methods is that their running cost due to the frequent calling of a large neural network model at inference time (as seen for the NEXT curve). We see in Figure5(d) that the wall clock time of using the GNN models is comparable to the standard heuristic-based BIT* and RRT*, when all algorithms can find paths. The main reason is that the

reduction in collision checking significantly reduces the overall time. We believe the GNNs can be further optimized to achieve faster inference as well.

In summary, experimental results show that the GNN-based approaches significantly reduce collision checking while maintaining high success rate, low path cost, and fast overall planning. The performance scales well from low-dimensional to high-dimensional problems.

6.2 Generalizability of Collision Reduction

A major challenge of learning-based approaches for planning is that small changes of the geometry of the environment can lead to abrupt change in the solutions, and thus lead the difficulty of generalization. In Figure6, we provide evidence that the GNN approach can alleviate this problem because of its access to the graph structures formed by samples uniformly taken from the space. In the 2D maze environment, the top one is easy while the bottom one is much harder, although the difference is just whether a narrow corridor is present to the left of the start state. For the UR5 with pads and poles, to solve the hard one, the robot arm needs to first rotate itself around the z-axis to bypass the small pads, and then rotate it back to fit the goal configuration. These problems are especially challenging for generalizing learned results to unseen environments.

We observe that the GNN-based components can handle the transition from the easy to hard problems consistently. In both environments, the GNN components find paths quickly, with 9 and 236 edges explored for 2D maze, and with 1 and 256 edges explored for UR5. In contrast, the NEXT model trained on the same training sets, can quickly find a path in the easy problem, but gets stuck in the hard one. Since the two problems are close to each other in the input space for NEXT, it is not surprising to see this difficulty of generalization. In fact, the only case where NEXT can eventually find a path on 2D Maze after more than 6K edge exploration is when the algorithm delegates 10% operations to standard RRT*. Without delegating to RRT*, NEXT gets stuck in local regions after 10K exploration steps on 2D and 1K steps on UR5.

6.3 Ablation Study: Probing Samples

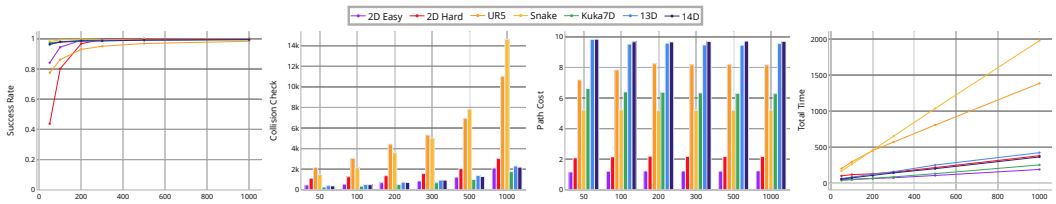

Figure 7: Comparison of performances for different probing samples from 2D to 14D problems.

We perform ablation studies of varying different parameters in our approach. *The full details are provided in the Appendix.* For instance, we use RGGs with different number of vertices from the free space, using 100, 200, 300, 500, 1000 samples, as illustrated in Figure 7. The success rate increases when there are more probing samples, which is consistent with the resolution-complete property of sampling-based planning. The path cost stays nearly the same for all settings, indicating robustness of the smoother models. The collision checks and planning time grows linearly with the probing samples. The reason is that the number of edges of k-NN RGG increments linearly with vertices, the input to the GNN grows linearly, thus the computation cost increases linearly on both CPU and GPU.

7 Conclusion

We presented a new learning-based approach to reducing collision checking in sampling-based motion planning. We train graph neural network (GNN) models to perform path exploration and path smoothing given the random geometric graphs (RGGs) generated from batch sampling. We rely on the ability of GNN for capturing important geometric patterns in graphs. The learned components can significantly reduce collision checking and improve overall planning efficiency in complex and high-dimensional motion planning environments.

Broader Impact

This work can be applied to mission-critical robotics applications such as autonomous driving, factory automation, and rescue robots. Some of these applications utilize sampling-based motion planning as a component, to generate a path which is used later for the controller to follow. The sampling-based methods are popular to provide a initial feasible path, but expensive when it comes to the collision checking, especially when the environment is complicated for simulation. The proposed method is fast, efficient, and probabilistic-complete, which is important to the reliability. The GNN explorer potentially requires fewer collision checks to find a feasible path, and generates a shorter path combined with the GNN smoother. It is possible for this work to be provided as a proxy to planning module, for efficient planning. On the other side, we believe the training data has important impact for the learning and the inference performance. The performance might degrade, when the scale of the graph at inference phase does not match with the training data.

Potential failures of the system can be caused by the difficulty to follow the planned path for the low-level controller, and by scenarios where the consumed energy is prioritized, or the data are expensive to collect. Future research can address these scenarios, and also situations which contain dynamic obstacles or kinodynamic constraints. In general, graph learning are being deployed to a wide range of applications. As most of the approaches which use neural networks might be, the learned GNN itself is a black box, and needs further study on safety, biases, and interpretability.

References

[1] J. Barraquand and J. Latombe. Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles. *Algorithmica*, 10(2-4):121–155, 1993. doi: 10.1007/BF01891837. URL `https://doi.org/10.1007/BF01891837`.

[2] M. J. Bency, A. H. Qureshi, and M. C. Yip. Neural path planning: Fixed time, near-optimal path generation via oracle imitation. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*, pages 3965–3972. IEEE, 2019. doi: 10.1109/IROS40897.2019.8968089. URL `https://doi.org/10.1109/IROS40897.2019.8968089`.

[3] X. Bresson and T. Laurent. The transformer network for the traveling salesman problem. *CoRR*, abs/2103.03012, 2021. URL `https://arxiv.org/abs/2103.03012`.

[4] J. Chase Kew, B. Ichter, M. Bandari, T.-W. E. Lee, and A. Faust. Neural collision clearance estimator for batched motion planning. In S. M. LaValle, M. Lin, T. Ojala, D. Shell, and J. Yu, editors, *Algorithmic Foundations of Robotics XIV*, pages 73–89, Cham, 2021. Springer International Publishing. ISBN 978-3-030-66723-8.

[5] B. Chazelle. Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm. *SIAM J. Comput.*, 13(3):488–507, 1984. doi: 10.1137/0213031. URL `https://doi.org/10.1137/0213031`.

[6] B. Chen, B. Dai, Q. Lin, G. Ye, H. Liu, and L. Song. Learning to plan in high dimensions via neural exploration-exploitation trees. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL `https://openreview.net/forum?id=rJgJDAVKvB`.

[7] M. Cherif. Kinodynamic motion planning for all-terrain wheeled vehicles. In *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*, pages 317–322. IEEE Robotics and Automation Society, 1999. doi: 10.1109/ROBOT.1999.769998. URL `https://doi.org/10.1109/ROBOT.1999.769998`.

[8] H. M. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, S. Thrun, and R. C. Arkin. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[9] H. Dai, Y. Li, C. Wang, R. Singh, P. Huang, and P. Kohli. Learning transferable graph exploration. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 2514–2525, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/afe434653a898da20044041262b3ac74-Abstract.html.

[10] N. Das and M. Yip. Learning-based proxy collision detection for robot motion planning applications. *IEEE Transactions on Robotics*, 36(4):1096–1114, 2020.

[11] R. Diankov and J. Kuffner. Randomized statistical path planning. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 29 - November 2, 2007, Sheraton Hotel and Marina, San Diego, California, USA*, pages 1–6. IEEE, 2007. doi: 10.1109/IROS.2007.4399557. URL https://doi.org/10.1109/IROS.2007.4399557.

[12] B. R. Donald, P. G. Xavier, J. F. Canny, and J. H. Reif. Kinodynamic motion planning. *J. ACM*, 40(5):1048–1066, 1993. doi: 10.1145/174147.174150. URL https://doi.org/10.1145/174147.174150.

[13] I. Drori, A. Kharkar, W. R. Sickinger, B. Kates, Q. Ma, S. Ge, E. Dolev, B. Dietrich, D. P. Williamson, and M. Udell. Learning to solve combinatorial optimization problems on real-world graphs in linear time. In M. A. Wani, F. Luo, X. A. Li, D. Dou, and F. Bonchi, editors, *19th IEEE International Conference on Machine Learning and Applications, ICMLA 2020, Miami, FL, USA, December 14-17, 2020*, pages 19–24. IEEE, 2020. doi: 10.1109/ICMLA51294.2020.00013. URL https://doi.org/10.1109/ICMLA51294.2020.00013.

[14] S. Emmons, A. Jain, M. Laskin, T. Kurutach, P. Abbeel, and D. Pathak. Sparse graphical memory for robust planning. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/385822e359afa26d52b5b286226f2cea-Abstract.html.

[15] B. Eysenbach, R. Salakhutdinov, and S. Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 15220–15231, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/5c48ff18e0a47baaf81d8b8ea51eec92-Abstract.html.

[16] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Bit*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. *CoRR*, abs/1405.5848, 2014. URL http://arxiv.org/abs/1405.5848.

[17] E. N. Gilbert. Random plane networks. *Journal of the society for industrial and applied mathematics*, 9(4):533–543, 1961.

[18] E. Heiden, L. Palmieri, S. Koenig, K. O. Arras, and G. S. Sukhatme. Gradient-informed path smoothing for wheeled mobile robots. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 1710–1717. IEEE, 2018. doi: 10.1109/ICRA.2018.8460818. URL https://doi.org/10.1109/ICRA.2018.8460818.

[19] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[20] B. Ichter and M. Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics Autom. Lett.*, 4(3):2407–2414, 2019. doi: 10.1109/LRA.2019.2901898. URL https://doi.org/10.1109/LRA.2019.2901898.

[21] B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 7087–7094. IEEE, 2018. doi: 10.1109/ICRA.2018.8460730. URL https://doi.org/10.1109/ICRA.2018.8460730.

[22] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[23] L. Janson and M. Pavone. Fast marching trees: a fast marching sampling-based method for optimal motion planning in many dimensions - extended version. *CoRR*, abs/1306.3532, 2013. URL http://arxiv.org/abs/1306.3532.

[24] P. Jiménez, F. Thomas, and C. Torras. Collision detection algorithms for motion planning. In *Robot motion planning and control*, pages 305–343. Springer, 1998.

[25] T. Jurgenson and A. Tamar. Harnessing reinforcement learning for neural motion planning. In A. Bicchi, H. Kress-Gazit, and S. Hutchinson, editors, *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, 2019. doi: 10.15607/RSS.2019.XV.026. URL https://doi.org/10.15607/RSS.2019.XV.026.

[26] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

[27] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Robotics Res.*, 30(7):846–894, 2011. doi: 10.1177/0278364911406761. URL https://doi.org/10.1177/0278364911406761.

[28] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6348–6358, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/d9896106ca98d3d05b8cbdf4fd8b13a1-Abstract.html.

[29] A. Khan, A. Ribeiro, V. Kumar, and A. G. Francis. Graph neural networks for motion planning. *CoRR*, abs/2006.06248, 2020. URL https://arxiv.org/abs/2006.06248.

[30] K. Kondo. Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration. *IEEE Trans. Robotics Autom.*, 7(3):267–277, 1991. doi: 10.1109/70.88136. URL https://doi.org/10.1109/70.88136.

[31] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. ISBN 9780511546877. doi: 10.1017/CBO9780511546877. URL http://planning.cs.uiuc.edu/.

[32] S. M. LaValle and J. J. K. Jr. Randomized kinodynamic planning. In *1999 IEEE International Conference on Robotics and Automation, Marriott Hotel, Renaissance Center, Detroit, Michigan, USA, May 10-15, 1999, Proceedings*, pages 473–479. IEEE Robotics and Automation Society, 1999. doi: 10.1109/ROBOT.1999.770022. URL https://doi.org/10.1109/ROBOT.1999.770022.

[33] L. Lee, E. Parisotto, D. S. Chaplot, E. P. Xing, and R. Salakhutdinov. Gated path planning networks. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2953–2961. PMLR, 2018. URL http://proceedings.mlr.press/v80/lee18c.html.

[34] Q. Li, F. Gama, A. Ribeiro, and A. Prorok. Graph neural networks for decentralized multi-robot path planning. *CoRR*, abs/1912.06095, 2019. URL http://arxiv.org/abs/1912.06095.

[35] K. M. Lynch and M. T. Mason. Stable pushing: Mechanics, controllability, and planning. *Int. J. Robotics Res.*, 15(6):533–556, 1996. doi: 10.1177/027836499601500602. URL https://doi.org/10.1177/027836499601500602.

[36] Z. S. O. B. . S. S. Madaan, R. Learning adaptive sampling distributions for motion planning by self-imitation. *Workshop on Machine Learning in Robot Motion Planning, IEEE IROS*, 2018.

[37] M. W. Mueller, M. Hehn, and R. D'Andrea. A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Transactions on Robotics*, 31(6):1294–1310, 2015.

[38] S. Niu, S. Chen, H. Guo, C. Targonski, M. C. Smith, and J. Kovacevic. Generalized value iteration networks: Life beyond lattices. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6246–6253. AAAI Press, 2018. URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16552`.

[39] S. M. Persson and I. Sharf. Sampling-based a* algorithm for robot path-planning. *Int. J. Robotics Res.*, 33(13):1683–1708, 2014. doi: 10.1177/0278364914547786. URL `https://doi.org/10.1177/0278364914547786`.

[40] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 77–85. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.16. URL `https://doi.org/10.1109/CVPR.2017.16`.

[41] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, 37 (1):48–66, 2021. doi: 10.1109/TRO.2020.3006716.

[42] J. H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 421–427, 1979. doi: 10.1109/SFCS.1979.10.

[43] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: bayesian personalized ranking from implicit feedback. In J. A. Bilmes and A. Y. Ng, editors, *UAI 2009, Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, Montreal, QC, Canada, June 18-21, 2009*, pages 452–461. AUAI Press, 2009. URL `https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1630&proceeding_id=25`.

[44] N. Savinov, A. Dosovitskiy, and V. Koltun. Semi-parametric topological memory for navigation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=SygwwGbRW`.

[45] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[46] D. Shah, B. Eysenbach, N. Rhinehart, and S. Levine. RECON: rapid exploration for open-world navigation with latent goal models. *CoRR*, abs/2104.05859, 2021. URL `https://arxiv.org/abs/2104.05859`.

[47] M. P. Strub and J. D. Gammell. Advanced bit* (abit*): Sampling-based planning with advanced graph-search techniques. In *2020 IEEE International Conference on Robotics and Automation, ICRA 2020, Paris, France, May 31 - August 31, 2020*, pages 130–136. IEEE, 2020. doi: 10.1109/ICRA40945.2020.9196580. URL `https://doi.org/10.1109/ICRA40945.2020.9196580`.

[48] R. A. M. Strudel, R. Garcia, J. Carpentier, J. Laumond, I. Laptev, and C. Schmid. Learning obstacle representations for neural motion planning. *CoRR*, abs/2008.11174, 2020. URL `https://arxiv.org/abs/2008.11174`.

[49] A. Tamar, S. Levine, P. Abbeel, Y. Wu, and G. Thomas. Value iteration networks. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2146–2154, 2016. URL https://proceedings.neurips.cc/paper/2016/hash/c21002f464c5fc5bee3b98ced83963b8-Abstract.html.

[50] H. Tang, Z. Huang, J. Gu, B.-L. Lu, and H. Su. Towards scale-invariant graph-related problem solving by iterative homogeneous gnns. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15811–15822. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/b64a70760bb75e3ecfd1ad86d8f10c88-Paper.pdf.

[51] M. Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936, 2015.

[52] P. Velickovic, L. Buesing, M. C. Overlan, R. Pascanu, O. Vinyals, and C. Blundell. Pointer graph networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/176bf6219855a6eb1f3a30903e34b6fb-Abstract.html.

[53] P. Velickovic, R. Ying, M. Padovano, R. Hadsell, and C. Blundell. Neural execution of graph algorithms. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SkgKO0EtvS.

[54] K. Xu, J. Li, M. Zhang, S. S. Du, K. Kawarabayashi, and S. Jegelka. What can neural networks reason about? In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=rJxbJeHFPS.

[55] F. Xue and P. R. Kumar. The number of neighbors needed for connectivity of wireless networks. *Wireless networks*, 10(2):169–181, 2004.

[56] Y. Yan, K. Swersky, D. Koutra, P. Ranganathan, and M. Hashemi. Neural execution engines: Learning to execute subroutines. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/c8b9abffb45bf79a630fb613dcd23449-Abstract.html.

[57] G. Yang, A. Zhang, A. S. Morcos, J. Pineau, P. Abbeel, and R. Calandra. Plan2vec: Unsupervised representation learning by latent plans. In A. M. Bayen, A. Jadbabaie, G. J. Pappas, P. A. Parrilo, B. Recht, C. J. Tomlin, and M. N. Zeilinger, editors, *Proceedings of the 2nd Annual Conference on Learning for Dynamics and Control, L4DC 2020, Online Event, Berkeley, CA, USA, 11-12 June 2020*, volume 120 of *Proceedings of Machine Learning Research*, pages 935–946. PMLR, 2020. URL http://proceedings.mlr.press/v120/yang20b.html.

[58] C. Zhang, J. Huh, and D. D. Lee. Learning implicit sampling distributions for motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 3654–3661. IEEE, 2018. doi: 10.1109/IROS.2018.8594028. URL https://doi.org/10.1109/IROS.2018.8594028.

[59] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4(4):3529–3536, 2019.

[60] Z. Zhu, E. Schmerling, and M. Pavone. A convex optimization approach to smooth trajectories for motion planning with car-like robots. In *2015 54th IEEE conference on decision and control (CDC)*, pages 835–842. IEEE, 2015.

[61] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] **See Broader Impact.**
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] **See Broader Impact.**
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] **See supplemental material.**
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] **See Section 4, 5, 6.1. and supplemental material.**
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] **See Figure 5 and supplemental material.**
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] **See supplemental material.**
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] **See Section 6.1.**
   (b) Did you mention the license of the assets? [Yes] **See Section 6.1.**
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] **Provided in supplemental material.**
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] **Codes from others were on GitHub in open source.**
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]