Learning-based Manipulation Planning in Dynamic Environments Using GNNs and Temporal Encoding

Anonymous Author(s)
Affiliation
Address
email

Abstract

Learning-based approaches have shown promising performance for improving the efficiency of motion planning in robot manipulation problems, but mostly in the setting of static environments. For the more challenging problem of manipulation planning in dynamic environments, such as for multi-arm assembly tasks or human-robot interaction, motion planners need to consider the trajectories of the dynamic obstacles, and reason about the temporal-spatial interactions between the ego-arm and the other objects. We propose a GNN-based neural architecture that involves temporal encoding, and use imitation learning with data aggregation procedures for learning both the embedding and edge prioritization policies. Experiments show that the learning-based approach can significantly accelerate online planning in comparison to state-of-the-art complete dynamic planning algorithms. The proposed methods can reduce costly collision checking operations by more than 1000x, thus reducing the online planning time by over 95%, while also achieving high success rate on hard instances.

1 Introduction

Motion planning for manipulation tasks has been a longstanding challenge in robotics [2, 10]. Learning-based approaches can exploit patterns in the configuration space to accelerate planning and have shown promising results [29, 5, 3]. Existing learning-based approaches typically combine reinforcement learning (RL) and imitation learning (IL) to learn policies for sampling or ranking the options at each step of the planning process [34, 45, 4]. Graph Neural Networks (GNNs) are a popular choice of representation for motion planning problems, because of their capability to capture geometric information and are invariant to the permutations of the sampled graph [19, 22, 23, 44].

Manipulation planning in dynamic environments, such as for multi-arm assembly tasks or safe manipulation around humans, is significantly more challenging than in static environments. Although learning-based approaches have shown success in static environments, it remains open whether they can also accelerate manipulation planning in dynamic environments. In dynamic environments, each dynamic obstacle produces trajectories in the temporal-spatial space, and the motion planner for the ego-arm needs to consider global geometric constraints in the configuration space at each time step (Figure 1). This dynamic nature of the environment generates a larger state space of sequences of graphs, which is also very sensitive to the change in one single dimension: time. A small change in the timing of the ego-robot or the obstacles, in two spatially similar patterns, may result in completely different planning problems. For instance, the dynamic obstacle may create a small time window for the ego-robot to pass through, and if that window is missed then the topology of configuration space may change globally. Consequently, we need to design special architectures that can both encode

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

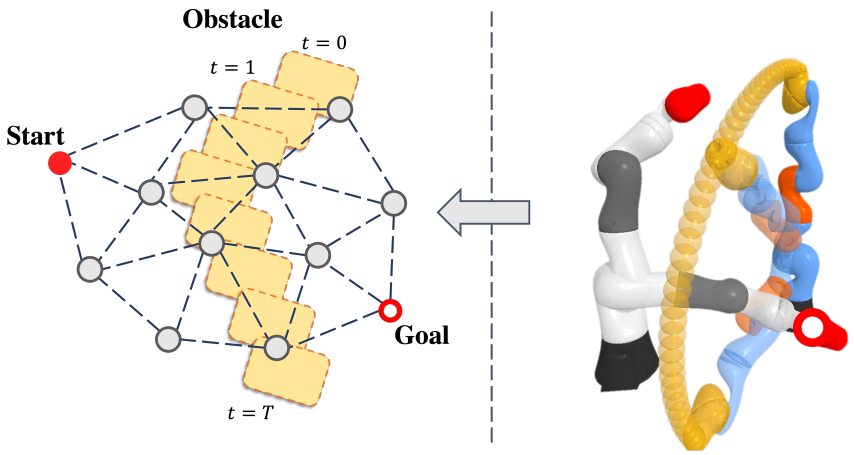

Figure 1: Left: A sampled graph from the configuration space. A dynamic obstacle, colored in yellow, moves from timestep $t = 0$ to $t = T$. The goal of our approach is to search for a path on the graph connecting the start to the goal, without any collisions at all the timesteps. Right: The ego-arm tries to avoid collision with the dynamic obstacle arm while reaching the goal.

the graph structures well, and also be able to robustly infer temporal information. Indeed, complete search algorithms for dynamic manipulation planning, such as variations of the leading method of Safe Interval Path Planning (SIPP) [32, 24], focus on reasoning about the temporal intervals that are safe for the ego-robot. The complete algorithms require significantly more computation and collision checking operations compared to the static setting, often making it impractical to consider dynamic manipulation planning problems in practice.

We propose a novel Graph Neural Network (GNN) architecture manipulation planning in dynamic environments. We follow the sampling-based motion planning methods [18, 17], where path planning is performed on a sampled graph from the configuration space.The architecture takes into account of the overall sampled graph in the configuration space, the obstacle's trajectory in the work space, and the current state of the ego-robot. The output is a vector of priority values on the candidate edges at the current state of the robot. The encoding is performed in two stages. In the first stage, we encode the graph structure information into the sampled graph using attention mechanisms [41], and also design a temporal encoding method for the obstacle trajectories. The temporal encoding uses the idea of positional encoding from the Transformer and Nerf [41, 26], which encourages the neural network to learn the temporal pattern from high-frequency input signals.In the second stage of encoding, we take into account of the ego-robot's current vertex in the configuration space, the local graph structure, and the current time-shifted trajectories of the obstacles. This two-stage structure is a significant extension of the previous use of GNNs in static environments [22, 23, 44] and it is important for making high-quality predictions on the priority values. The entire GNN of both stages will be trained simultaneously in an end-to-end fashion.

Because of the complexity of the GNN architecture, we found that RL-based methods can not train generalizable models based on the architecture, and imitation learning with data aggregation (DAgger) is the key. We utilize the complete dynamic planning algorithm SIPP [32] as the expert to first perform behavior cloning as a warm-up training, and then allow the ego-robot to self-explore and learn from the expert following the DAgger method [36, 11].

We evaluate the proposed approach in various challenging manipulation environments ranging from 2-DoF arms to 7-DoF KUKA arms. Experiments show that our method significantly reduces collision checking by more than 1000x compared to the complete algorithm SIPP, which leads to reducing the online computation time by over 95%, and also achieves high success rate compared to other learning and heuristic baselines.

2 Related Work

Manipulation Planning in Dynamic Environments. The planning in dynamic environment has proven to be difficult [35]. The work of [32] and its variations [9, 28, 24] are the leading framework for manipulation planning in dynamic environments. The method significantly reduces the temporal-spatial search space by introducing a new data structure by grouping safe configurations over a period of time *safe intervals*, [32]. Based on this data structure, Safe Interval Path Planning (SIPP) can find the optimal paths on a configuration graph. Further improvements are made to improve the efficiency of SIPP by leveraging state dominance [9] or render the any-time search feature [28]. Recently, [24] shows SIPP is also powerful in planning the paths of high-dimensional manipulators on configuration roadmaps [18] with the presence of dynamic obstacles, which is the problem of interests in this paper.

Many other algorithms consider path planning in dynamic environments for mobile robots [12, 31, 6]. Such problems are typically more efficiently solvable because of the low dimensionality and the use of more conservative planning methods. It is common to bound the robot to avoid the occupied space of moving obstacles for the entire or receding horizons.

Learning-based Motion Planning. Learning-based approaches typically consider motion planning as a sequential decision-making problem, which could be naturally tackled with reinforcement learning or imitation learning. With model-based reinforcement learning, DDPG-MP [16] integrates the known dynamic of robots and trains a policy network. [38] improves the obstacle encoding with the position and normal vectors. OracleNet [1] learns via oracle imitation and encodes the trajectory history by an LSTM [13]. Other than greedily predicting nodes that sequentially form a trajectory, various approaches have been designed to first learn to sample vertices, then apply search algorithms to the sampled graphs [14]. [15, 34, 45] learn sampling distributions to generate critical vertices from configuration space. Inductive bias is also leveraged by several works, including [25, 4, 21], which design neural networks that take structured inputs. Existing learning-based approaches have also considered improving collision detection [5, 3, 44] and learning to explore edges given fixed graphs [39, 21]. For non-static environments, most works focus on multi-agent scenarios [30, 11], where there do not exist non-cooperative dynamic obstacles.

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. For motion planning, [19] utilizes GNN to identify critical samples. [22, 23, 7] predicts the action for each agent in the grid-world environment using graph neural networks. [40, 20] learns the control policy for each robot in a large-scale swarm. [46] learns the submodular aciton selection for continuous space. [44] learns a GNN-based heuristic function for search in static environment with completeness guarantee.

3 Preliminaries

Sampling-based Motion Planning with Dynamic Obstacles. We focus on the sampling-based motion planning, in which we sample from the *configuration space* $C \subseteq \mathbb{R}^n$ where n is the number of degree-of-freedom for the ego-robot. All the sampled vertices form a graph $G = \langle V, E \rangle$, with start vertex v_s and goal vertex v_g. The connectivity in the graph is determined by r-disc or k-nearest-neighbor (k-NN) methods[8, 43]. In the setting of manipulation planning, we assume access to the full trajectories of the dynamic obstacles, which are often the other robot arms. We represent the trajectory of dynamic obstacles in the *workspace* as the vector of all the joint positions in time window $T \subseteq \mathbb{R}^{\geq 0}$. The goal of the manipulation planning problem is to find a path from v_s to v_g in the sampled graph that is free of collision with the dynamic obstacles at all time steps in T.

Graph Neural Networks (GNNs). GNNs can efficiently learn the representation of each vertex and edge on a graph $G = \langle V, E \rangle$ by message passing from neighbors. With MLP networks f and g, GNN encodes the representation $h_i^(k+1)$ of vertex v_i after k aggregations as

$$h_i^{(k+1)} = g(h_i^{(k)}, \oplus^{(k)}(\left\{ f(h_i^{(k)}, h_j^{(k)}) \mid (v_i, v_j) \in E \right\})) \tag{1}$$

where the initial data of vertex v_i is x_i and $h_i^{(1)} = x_i$. $\oplus$ is the permutation-invariant aggregation function on sets, such as mean, max or sum.

Attention Mechanisms. For the purpose of introducing the dynamic obstacles in our network, while reasoning about the temporal-spatial interactions between the ego-arm and the obstacles, we use attention mechanism to encode the obstacle features. In the general form of attention mechanism, there are n keys each with dimension d_q: $K \in \mathbb{R}^{n \times d_k}$, each key has to a value $V \in \mathbb{R}^{n \times d_v}$. Given m query vectors $Q \in \mathbb{R}^{m \times d_k}$, we use a typical attention function $\mathbf{Att}(K, Q, V)$ for each query as $\mathbf{Att}(K, Q, V) = \text{softmax}(QK^T/\sqrt{d_k})V$ [41].

Imitation Learning. Imitation learning aims to provide guidance to train policy without explicitly designing reward functions. Our model also benefits from this technique which follows the demonstration by the oracle and learns mappings from the observations to actions. Specifically, in the task of sampling-based motion planning, we follow the oracle by predicting the priority value of subsequent edges and prioritize the one given by the oracle. However, imitation learning can result in poor performance in practice, in the cases that the ego-arm can stray from the right path and cannot recover due to the lack of enough guidance from the oracle. To this end, imitation with data aggregation (DAgger) [36] provides an iterative algorithm to generate new data by querying the oracle after executing the current policy, then trains a new policy after aggregating the dataset.

4 GNN-TE: Dynamic Motion Planning with GNNs and Temporal Encoding

4.1 Overall Architecture

Our dynamic motion planning network **GNN-TE** leverages the graph and temporal nature of the dynamic motion planning task. It consists of two stages. The first stage is the global GNN encoder $\mathcal{N}_G$ which encodes the global information about the ego-arm and obstacles in the environment. The second stage is the local planner $\mathcal{N}_p$ which executes motion planning utilizing the encoding output of the first stage. Fig 2 shows the two-stage architecture of our network.

First-stage global GNN Encoder $\mathcal{N}_G$ The GNN encoder $\mathcal{N}_G$ takes in a sampled random geometric graph $G = \langle V, E \rangle$, $V = \{v_s, v_g, v\}$. For an n-dimensional configuration space, each vertex $v_i \in \mathbb{R}^{n+1}$ contains an n-dimensional configuration component and a 1-dimensional one-hot label indicating if it is the special goal vertex.

The vertices and the edges are first encoded 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 encoding. Specifically, to encode the graph while informing the network of the task specific goal, we choose $x = h_x(v, v_g, v - v_g, ||v - v_g||_2^2) y = h_y(v_i, v_j, v_j - v_i)$, where h_x and h_y are two different two-layer MLPs. The L2 distance to the goal $||v - v_g||_2^2$ serves as the heuristic information for $\mathcal{N}_G$.

The dynamic obstacles O form barriers on the top of the graph G, and we incorporate the trajectories of them to $\mathcal{N}_G$ to leverage the global environment and assist the planning of $\mathcal{N}_P$. Additionally, to inform the networks about the relative time over the horizon, we incorporate **temporal encoding** with the obstacles, i.e. $O = O + TE$, which we will discuss in the later section. With the sequential property of the trajectory, we use the attention mechanism to model the temporal-spatial interactions between the ego-arm and the obstacles. Concretely, the obstacles are encoded into the vertex and edge of G as:

$$x = x + \mathbf{Att}(f_{K_x^{(i)}}(O), f_{Q_x^{(i)}}(x), f_{V_x^{(i)}}(O)) \tag{2}$$

$$y = y + \mathbf{Att}(f_{K_y^{(i)}}(O), f_{Q_y^{(i)}}(y), f_{V_y^{(i)}}(O)) \tag{3}$$

Taking the vertex and edge encoding x, y, the GNN $\mathcal{N}_G$ aggregates the local information for each vertex and edge from the neighbors, by performing the following operation with 2 two-layer MLPs f_x and f_y:

$$\begin{aligned} x_i &= \max\left(x_i, \max\{f_x(x_j - x_i, x_j, x_i, y_l) \mid e_l : \langle v_i, v_j \rangle \in E\}\right), \forall v_i \in V \\ y_l &= \max(y_l, f_y(x_j - x_i, x_j, x_i)), \forall e_l : \langle v_i, v_j \rangle \in E \end{aligned} \tag{4}$$

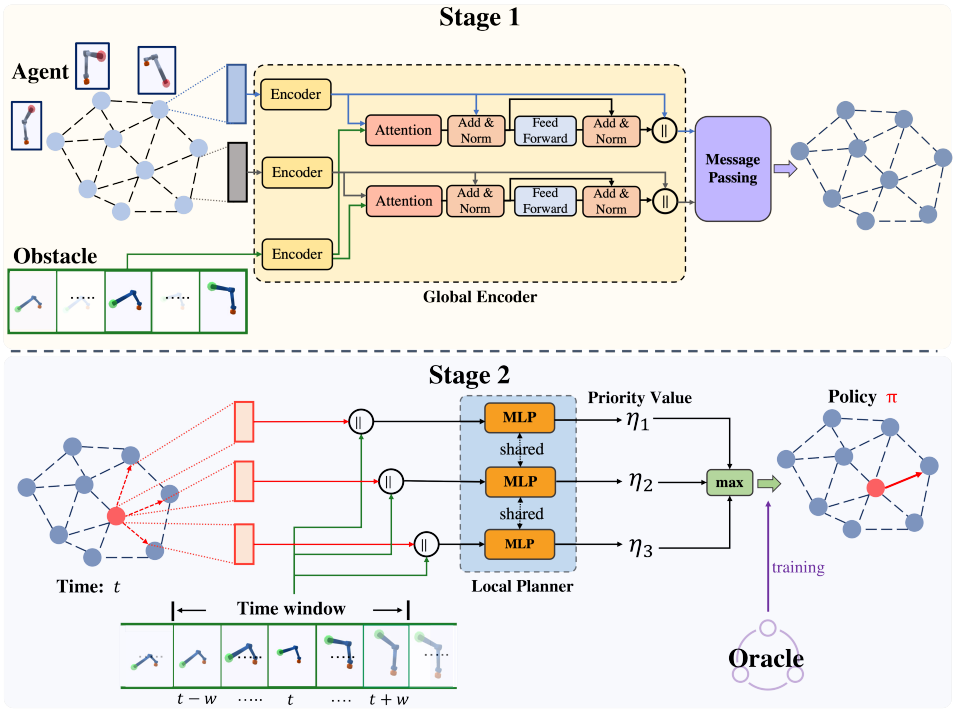

Figure 2: Overall two-stage architecture of the proposed **GNN-TE**. At stage 1, we use the global encoder to encode the global information of the ego-arm and the obstacles leveraging the attention mechanism. Then the model will output the encoding of each edge after the message passing. At stage 2, the local planner will take in the edge encoding output of the stage 1, along with the obstacle encoding within a time window, to predict the priority value of each outgoing edge. The planner will propose the edge with the highest priority value to take as the output policy.

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 [33]. The edge information is also incorporated into the vertex by adding y_l as the input to f_x. Also because the Equation 4 is a homogeneous function that updates on the x and y in a self-iterating way, we can update without introducing redundant layers over multiple loops. After several iterations, the first-stage $\mathcal{N}_G$ outputs the encoding of each vertex x_i and edge y_l.

Second-stage local planner $\mathcal{N}_P$ After the GNN $\mathcal{N}_G$ encodes the information of the configuration graph and obstacles, the second-stage local planner $\mathcal{N}_P$ utilizes the encoding and performs motion planning. Specifically, when arriving at a vertex v_i at time t_i, $\mathcal{N}_P$ predicts the priority value η_j of all the subsequent edges $e_j \in E_j$ with the expression $\eta_j = g(y_l, O_{t_i-w}, O_{t_i-w+1}, ..., O_{t_i+w-1}, O_{t_i+w})$, where g is a multi-layer MLP. Note that apart from the encoding of the subsequent edge, we also input the local obstacle encoding within a time window w of the current arrival time t_i. This serves as the local-level information for the network $\mathcal{N}_P$ to plan towards the goal vertex while considering the barriers of dynamic obstacles to avoid collisions. In the inference time, we use $\mathcal{N}_P$ to choose the edge with the highest priority value while keeping track of the current time.

4.2 Temporal Encoding

Spatial encoding is a crucial design in the Transformer architecture [42] for making use of the order of the sequence. Dynamic motion planning requires the models to infer the relative position of obstacles and how they interact with the ego-arm at each time step. So along with the positions of the obstacles in the workspace, we add temporal encoding $TE \in \mathbb{R}^{T \times d_{TE}}$ at time t with k-th dimension

$$TE(t, 2k) = \sin(\omega^{-2k/d_{TE}} t) \text{ and } TE(t, 2k+1) = \cos(\omega^{-2k/d_{TE}} t) \tag{5}$$

where $\omega \in \mathbb{Z}^+$ is a fixed frequency. We choose the temporal encoding to have the same dimension as our obstacle encoding and add them together before input into the networks $\mathcal{N}_G$ and $\mathcal{N}_P$ as the obstacle encoding. We illustrate the overall encoding procedure on the left of Figure 3.

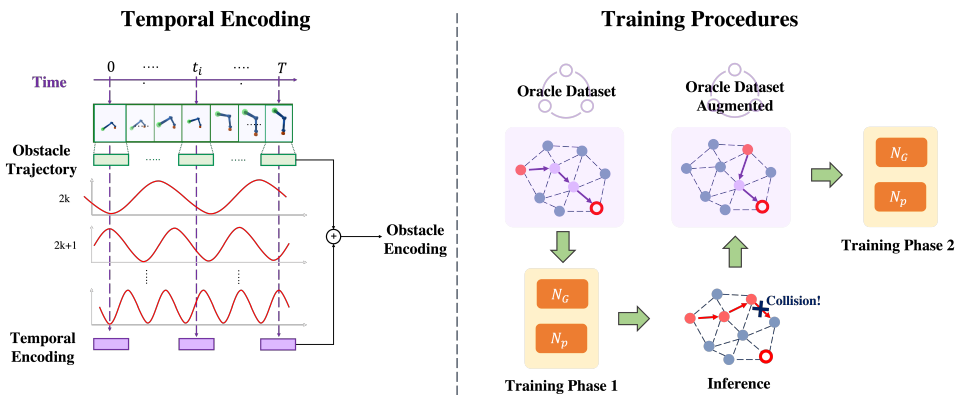

Figure 3: Left: Temporal encoding is incorporated when representing the dynamic obstacle sequence. Right: Training Procedures with DAgger. The proposed GNN is first trained to imitate an optimal oracle, and then improves itself by self-exploring the data with feedback from the oracle.

4.3 Training and Inference Procedures

In each training problem, along with the dynamic obstacles O, start vertex v_s and goal vertex v_g, we sample a k-NN graph $G = \langle V, E \rangle, V = \{v_s, v_g, v\}$, where v is the vertices sampled from the configuration space of the ego-arm. At the first stage, we use the global GNN encoder $\mathcal{N}_G$ to encode the graph and dynamic obstacles, and the local planner $\mathcal{N}_p$ in the second stage uses the encoding as the input to predict the priority value η of the subsequent edges.

Imitation from SIPP with Data Aggregation We train our two-stage network $\mathcal{N}_G$ and $\mathcal{N}_p$ in an end-to-end manner by imitating an oracle. Specifically, we use Safe Interval Path Planning (SIPP) [32] to compute the shortest non-collision motion path and use it as the oracle.

In the first stage, $\mathcal{N}_G$ will process the graph and the obstacle trajectories and output the encoded features of each vertex and edge. Then in the second stage, we train the networks to imitate the oracle SIPP along the optimal path. Concretely, starting from the v_s, SIPP provides the optimal path $\pi^* = \{(v_i^*, t_i^*)\}_{i \in [0,n]}$ with the vertex v_i^* and the corresponding arrival time t_i^*. When arriving at the vertex v_i^* at time t_i^*, our local planner $\mathcal{N}_p$ will input the edge feature y_l along with the obstacle encoding in the time window $[t_{i-w}^*, t_{i+w}^*]$ to predict the priority value of all the subsequent edges E_i. Then it prioritizes the next edge on the optimal path $e_i^* : \langle (v_i^*, t_i^*), (v_{i+1}^*, t_{i+1}^*) \rangle$ among E_i. We maximize the priority value $\eta_{e_i^*}$ of e_i^* over all other $e_i \in E_i \setminus \{e_i^*\}$ with the standard cross entropy loss $L_i = -\log(\exp(\eta_{e_i^*})/(\Sigma_{e_i \in E_i} \exp(\eta_{e_i})))$.

Since SIPP only provides the guidance on the optimal path, when the planned path of $\mathcal{N}_p$ deviates from the optimal path, our network cannot sufficiently imitate the oracle. To this end, we use DAgger [36] to encourage the network to learn from these sub-optimal paths. We first train our network for k iterations with pure demonstrations from SIPP. Then we explore a path π^k on the graph using the priority value predicted by the current network, which may not reach the goal vertex v_g or be optimal. We randomly stop at the vertex v_i^k at time t_i^k and query the oracle. SIPP treats v_i^k and t_i^k as the start vertex and the start time respectively, along with the obstacles trajectory starting at t_i^k, calculates the optimal path. The new demonstrations are aggregated to the previous dataset to keep training the network.

Inference with Time Tracking. Given a graph $G = \langle V, E \rangle, V = \{v_s, v_g, v\}$, with the trajectories of obstacles O, $\mathcal{N}_G$ will first encode the graph and obstacles. Next, $\mathcal{N}_P$ executes motion planning by predicting the priority values and follow the edge with the maximum one, while keeping track of the arrival time at each vertex, $e_{\pi_i} = \text{argmax}_{e_i \in E_i} \mathcal{N}_P(\eta_i | V, E, O, \mathcal{N}_G, t_i)$. Note that e_{π_i} is also

conditioned on the current arrival time t_i. The priority value η_i is the output of $\mathcal{N}_P$ when we arrive at vertex v_i at time t_i. After the edge e_{π_i} is proposed by the network, we check the collision on e_{π_i} started at t_i. If there is no collision, we add (e_{π_i}, t_i) into the current path π, otherwise, we query $\mathcal{N}_P$ for another proposed edge with the next greater priority value. The planning will end if we succeed in finding a path π from v_s to v_g or fail when stopping at a vertex with all the subsequent edges with collisions. Optionally, when the latter one happens, we can backtrack to the last step and query for the top-5 greatest priorities in turn. Further discussions are covered in the experiments section.

5 Experiments

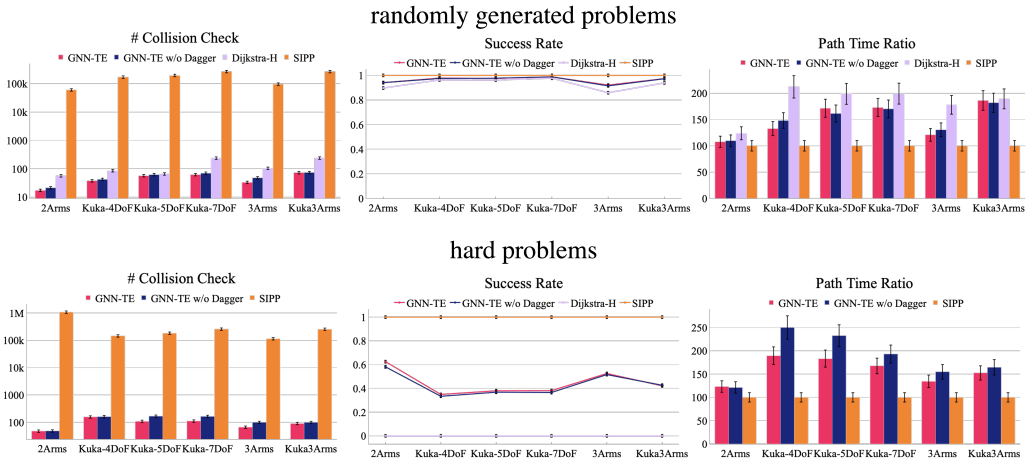

Figure 4: Overall Performance of baselines on collision checking, success rate and path time ratio. Our approach significantly reduces collision checking from 3000x to 4000x compared to the complete algorithm SIPP, improves the overall planning efficiency, and achieves high success rate.

Experiment Setup. We evaluate the proposed methods on various multi-arm assembly tasks where we manipulate the ego-arm and avoid the collisions with other obstacles, including moving arms and static obstacles. Specifically, the environments cover arms of 2 to 7 Degree of Freedom and 1 to 3 moving obstacle arms, including (i) 2Arms: 1 obstacle arm with 2 DoF. (ii) Kuka-4DoF: 1 obstacle arm with 4 DoF. (iii) Kuka-5DoF: 1 obstacle arm with 5 DoF. (iv) Kuka-7DoF: 1 obstacle arm with 7 DoF. (v) 3Arms: 2 obstacle arms with 2 DoF. (vi) Kuka3Arms: 2 obstacle arms with 7 DoF. Without loss of generality, we assume all the robot arms in an environment are of the same DoF and move at the same and constant speed. The static obstacles are represented as cuboids in the experiments.

Baselines. We compare our method **GNN-TE** with the lazy sample-based dynamic motion planning method **Dijkstra-Heuristic** (**Dijkstra-H**). It prioritizes the edges based on the shortest distance to the goal on the graph at each time step. We also compare our method with the oracle **SIPP** [32], which is a search algorithm that generates optimal paths using safe intervals.

Datasets. We randomly generate 2000 problems for training and 1000 problems for testing. In each problem, we randomly generate fixed-length trajectories of the moving obstacle arms and 1000 samples in the configuration space of the ego-arm that are collision-free with the static obstacles and provide them to the oracle to generate demonstration trajectories. In order to further investigate the performance of algorithms in the challenging environments, we also generate 1000 hard problems, where **Dijkstra-H** fails to find feasible paths.

Training and Testing Details. We first train the **GNN-TE** on all the training problems for 200 epochs. Afterward, we generate 1000 new training data with DAgger, and trained for another 100 epochs. We test all the algorithms on the provided graphs. We randomly split the test problems into 5 groups, and calculate the variance of the performance.

Evaluation Metrics. We measure the average **collision checking** of the common success cases to evaluate the effectiveness of algorithms in avoiding obstacles. Collision checking is the more expensive computation in motion planning, and reducing it can lead to a significant acceleration of online computation. We also report the average **success rate** over all the testing problems for all the algorithms. A trajectory is successful, only if it does not collide with any obstacles, and eventually reaches the goal given a limited time horizon. We also measure the optimality of the methods by comparing the **path time ratio** compared to the optimal paths found by SIPP on each success case. Note that the path time metric only measures the cost of the planned path, and does not include the online computation time needed.

5.1 Overall Performance

Figure 4 shows the overall performance of the algorithms in various environments, including all the baselines and **GNN-TE** without DAgger, i.e. pure imitation learning from the oracle.

In all the environments, **SIPP** gives the optimal complete non-collision path, but it suffers from the excessive amount of collision checking. **GNN-TE** significantly reduces the collision checking by more than 1000x, which corresponds to reducing online computation time by over 95%. At the same time, the methods have high success rates and better path time ratio compared to using simpler heuristics.

Fig. 5 shows the performance snapshots of the algorithms on a 2Arms and a Kuka-7DoF test case. In both cases, our method successes in planning a near-optimal path compared to the oracle **SIPP** whereas **Dijkstra-H** fails.

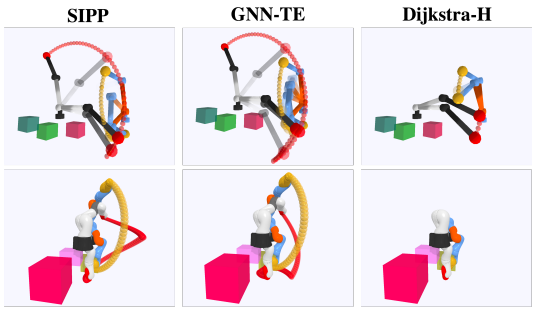

Figure 5: Snapshots of the trajectories of the output path on 2 test cases. The environments are 2Arms (first row) and Kuka-7DoF (second row).The ego-arm is of black and white with the red end-effector. The obstacle arm is of blue and orange with the yellow end-effector. In both environments, **Dijkstra-H** fails to find a path, while our method can yield near-optimal path compared to the oracle **SIPP**.

Performance on randomly generated test problems. As shown in Fig.4, our methods significantly reduces the collision checking by over 1000x compared to **SIPP**, i.e. 60k, 97k, 171k, 197k, 269k, 269k to 17.24, 33.08, 37.36, 56.89, 61.99, 72.63 on 2Arms, 3Arms, Kuka-4DoF, Kuka-5DoF, Kuka-7DoF, Kuka3Arms respectively. Because **SIPP** needs to search in both space and time dimensions, it suffers from a large number of collision checking. Our approach benefits from learning and achieves a minimal amount of collision checking even in the dynamic setting while not compromising much of the success rate, in which case our method outperforms **Dijkstra-H** on all the environments with higher success rates. Note that as a crucial part of the training procedures, DAgger also improves the performance in all environments.

Performance on hard test problems. On the hard test problems, **Dijkstra-H** fails to find feasible paths in the dynamic environment. Comparatively, our **GNN-TE** can successfully find solutions to these problems with considerable success rate and acceptable path time ratio. It is also worth noting that DAgger can better assist to improve the performance in the more challenging scenarios compared to the random generated problems.

Optional backtracking search. In the Appendix, we also report the result of **GNN-TE** and **Dijkstra-H** and them with backtracking **GNN-TE w. BT** and **Dijkstra-H w. BT** on 2Arms. On the same algorithm, the optional backtracking search will only result in a higher success rate while not affecting path time ratio and collision checking on the common success cases. The result of **Dijkstra-H w. BT** shows that although it improves the success rate significantly but sacrificing a tremendous number of collision checking. Nevertheless, our method outperforms the heuristic method on both with or without the backtracking search.

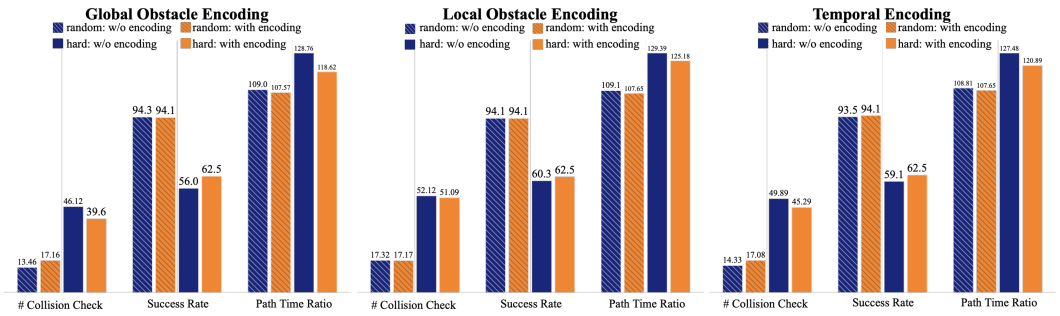

Figure 6: Ablation studies on 2Arms of (1) global obstacle encoding, (2) local obstacle encoding, and (3) temporal encoding. We have demonstrated that all these three components improve the effectiveness of the proposed approach. See Section 5.2 for more details.

Comparison with End-to-End RL. We also compare our approach with RL approaches, including DQN [27] and PPO [37]. The neural network architectures for these two baselines are implemented with the same GNN architectures as ours. We observe that even in 2Arms, the average success rate of DQN on the training set is only around 54%, while PPO only has a success rate of around 20%. They fail to find plans in randomly sampled test cases. We provide more details in the Appendix.

5.2 Ablation Study

We perform ablation studies on 2Arms of the different encoding in our model, including global and local obstacle encoding and temporal encoding. The results are shown in Fig. 6.

Global Obstacle Encoding. At stage 1, we both encode the configuration graph and the global obstacle trajectories with GNN-based global encoder. To investigate the effectiveness of leveraging global obstacle information, we conduct the experiment that we input the sampled configurations at stage 1 and only introduce the obstacle information at stage 2. As we can observe in the figure that although the path time ratio on the random problems slightly increases, the performance of path time ratio, success rate and collision checking improves in other scenarios. This results from the model receiving the overall trajectories of the obstacles, and the encoding helps to reduce collision checking and improves success rate especially on the hard problems.

Local Obstacle Encoding. We compare our model with the one omitting the local obstacle encoding at stage 2. It only inputs the temporal encoding corresponding to the arrival time as the time indicator for planning. The result has shown that the local obstacle encoding within a time window directly helps the local planner perform better.

Temporal Encoding. We analyze the importance of temporal encoding of time, where we remove it from both two stages and only input the trajectory of obstacles in the models. The results also show that temporal information helps **GNN-TE** making use of the order of the trajectory sequence on both random and hard problems.

6 Discussion and Conclusion

We proposed a GNN-based neural architecture **GNN-TE** for motion planning in dynamic environments. Temporal encoding is an important part of the GNN. We use imitation learning with DAgger for learning both the embedding and edge prioritization policies. We evaluate the proposed approach on various environments, ranging from 2-DoF arms to 7-DoF KUKA arms. Experiments show that the proposed approach can reduce costly collision checking operations by more than 1000x and reduces online computation time by over 95%. Future steps in this direction can involve using ideas from the complete planning algorithms, such as incorporating safe intervals, to improve the success rate on hard instances, as well as more compact architectures for further reducing online computation.

References

[1] 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.

[2] D. Berenson, S. S. Srinivasa, and J. J. Kuffner. Task space regions: A framework for pose-constrained manipulation planning. *Int. J. Robotics Res.*, 30(12):1435–1460, 2011. doi: 10.1177/0278364910396389. URL https://doi.org/10.1177/0278364910396389.

[3] 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.

[4] 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.

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

[6] J. Dong, M. Mukadam, F. Dellaert, and B. Boots. Motion planning as probabilistic inference using gaussian processes and factor graphs. In *Robotics: Science and Systems*, volume 12, 2016.

[7] F. Gama, Q. Li, E. I. Tolstaya, A. Prorok, and A. Ribeiro. Synthesizing decentralized controllers with graph neural networks and imitation learning. *IEEE Trans. Signal Process.*, 70:1932–1946, 2022. doi: 10.1109/TSP.2022.3166401. URL https://doi.org/10.1109/TSP.2022.3166401.

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

[9] J. P. Gonzalez, A. Dornbush, and M. Likhachev. Using state dominance for path planning in dynamic environments with moving obstacles. In *2012 IEEE International Conference on Robotics and Automation*, pages 4009–4015. IEEE, 2012.

[10] C. Guan, W. Vega-Brown, and N. Roy. Efficient planning for near-optimal compliant manipulation leveraging environmental contact. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 215–222. IEEE, 2018. doi: 10.1109/ICRA.2018.8462696. URL https://doi.org/10.1109/ICRA.2018.8462696.

[11] H. Ha, J. Xu, and S. Song. Learning a decentralized multi-arm motion planner. In J. Kober, F. Ramos, and C. J. Tomlin, editors, *4th Conference on Robot Learning, CoRL 2020, 16-18 November 2020, Virtual Event / Cambridge, MA, USA*, volume 155 of *Proceedings of Machine Learning Research*, pages 103–114. PMLR, 2020. URL https://proceedings.mlr.press/v155/ha21a.html.

[12] K. Hauser. On responsiveness, safety, and completeness in real-time motion planning. *Autonomous Robots*, 32(1):35–48, 2012.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics Autom.*, 12(4):566–580, 1996. doi: 10.1109/70.508439. URL https://doi.org/10.1109/70.508439.

[19] 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.

[20] A. Khan, E. Tolstaya, A. Ribeiro, and V. Kumar. Graph policy gradients for large scale robot control. In *Conference on robot learning*, pages 823–834. PMLR, 2020.

[21] 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.

[22] 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.

[23] Q. Li, W. Lin, Z. Liu, and A. Prorok. Message-aware graph attention networks for large-scale multi-robot path planning. *IEEE Robotics Autom. Lett.*, 6(3):5533–5540, 2021. doi: 10.1109/LRA.2021.3077863. URL https://doi.org/10.1109/LRA.2021.3077863.

[24] S. Li and J. A. Shah. Safe and efficient high dimensional motion planning in space-time with time parameterized prediction. In *2019 international conference on robotics and automation (ICRA)*, pages 5012–5018. IEEE, 2019.

[25] 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.

[26] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[28] V. Narayanan, M. Phillips, and M. Likhachev. Anytime safe interval path planning for dynamic environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4708–4715. IEEE, 2012.

[29] 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`.

[30] K. Okumura, R. Yonetani, M. Nishimura, and A. Kanezaki. Ctrms: Learning to construct cooperative timed roadmaps for multi-agent path planning in continuous spaces. In P. Faliszewski, V. Mascardi, C. Pelachaud, and M. E. Taylor, editors, *21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2022, Auckland, New Zealand, May 9-13, 2022*, pages 972–981. International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2022. URL `https://www.ifaamas.org/Proceedings/aamas2022/pdfs/p972.pdf`.

[31] C. Park, J. Pan, and D. Manocha. Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[32] M. Phillips and M. Likhachev. Sipp: Safe interval path planning for dynamic environments. In *2011 IEEE International Conference on Robotics and Automation*, pages 5628–5635. IEEE, 2011.

[33] 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`.

[34] 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.

[35] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. *Journal of the ACM (JACM)*, 41(4):764–790, 1994.

[36] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In G. J. Gordon, D. B. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 627–635. JMLR.org, 2011. URL `http://proceedings.mlr.press/v15/ross11a/ross11a.pdf`.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[38] 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`.

[39] 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`.

[40] E. I. Tolstaya, F. Gama, J. Paulos, G. J. Pappas, V. Kumar, and A. Ribeiro. Learning decentralized controllers for robot swarms with graph neural networks. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pages 671–682. PMLR, 2019. URL `http://proceedings.mlr.press/v100/tolstaya20a.html`.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. 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 5998–6008, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html`.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

[44] C. Yu and S. Gao. Reducing collision checking for sampling-based motion planning using graph neural networks. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 4274–4289, 2021. URL `https://proceedings.neurips.cc/paper/2021/hash/224e5e49814ca908e58c02e28a0462c1-Abstract.html`.

[45] 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`.

[46] L. Zhou, V. D. Sharma, Q. Li, A. Prorok, A. Ribeiro, and V. Kumar. Graph neural networks for decentralized multi-robot submodular action selection. *arXiv preprint arXiv:2105.08601*, 2021.

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]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (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]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (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)? [N/A]
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? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (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]