DynGMP: Graph Neural Network-based Motion Planning in Unpredictable Dynamic Environments

Wenjin Zhang[1], Xiao Zang[1], Lingyi Huang[1], Yang Sui[1], Jingjin Yu[2], Yingying Chen[1] and Bo Yuan[1,*]

***Abstract*— Neural networks have already demonstrated attractive performance for solving motion planning problems, especially in static and predictable environments. However, efficient neural planners that can adapt to unpredictable dynamic environments, a highly demanded scenario in many practical applications, are still under-explored. To fill this research gap and enrich the existing motion planning approaches, in this paper, we propose DynGMP, a graph neural network (GNN)-based planner that provides high-performance planning solutions in unpredictable dynamic environments. By fully leveraging the prior exploration experience and minimizing the replanning cost incurred by environmental change, DynGMP achieves high planning performance and efficiency simultaneously. Empirical evaluations across different environments show that DynGMP can achieve close to 100% success rate with fast planning speed and short path cost. Compared with existing non-learning and learning-based counterparts, DynGMP shows very significant planning performance improvement, e.g., at least $2.7\times$, $2.2\times$, $2.4\times$ and $2\times$ faster planning speed with low path distance in four environments, respectively.**

I. Introduction

Motion planning is a fundamental robotic task toward finding a collision-free path from a start state to an end state within the robot's free configuration space. Considering that robots in the real world typically operate in dynamic environments, in practice, deployed motion planners are typically required to provide strong *dynamic planning* capability to ensure the fast adjustment of the planned trajectory preserving high path quality. To that end, several dynamic motion planners, such as ERRT, Dynamic RRT*, and DRRT [1][2][3], have been proposed to operate in rapidly changing environments. However, as variants of conventional sampling-based planners (e.g., RRT [4], PRM [5], and RRT* [6]), existing dynamic planners naturally inherit a challenging limitation – the high computational costs incurred by the sequential and expensive invocations of sampling and collision-check sub-routines. Even worse, because obstacles in the dynamic environments often move randomly and unpredictably [7][3], the replanning process of existing dynamic planners require much more sampling and collision-checking calls to deal with high levels of uncertainty, further increasing the computational burden.

Recently, motivated by the widespread adoption of deep neural networks in many AI applications, learning-based motion planners have attracted much attention. By leveraging powerful learning and representation capabilities of neural networks, the state-of-the-art neural planners [8][9][10] can directly learn the suitable planning heuristics/policies from data, leading to superior performance than their non-learning-based counterparts in the static environment. Inspired by these encouraging successes, performing dynamic planning in a data-driven way, by its nature, is an apparently attractive solution. Unfortunately, to date, very few efforts have been reported toward designing neural dynamic planners. To our knowledge, only [11] proposes a graph neural network (GNN)-based dynamic motion planning solution. However, its approach is built upon the assumption that the movement of obstacles is fully predictable, making it less practical in real-world environments. Overall, high-performance motion planning for unpredictable dynamic environments is still under-explored and calls for an efficient solution.

To respond to this important demand, in this paper, we propose DynGMP, a GNN-based motion planner that can work in practical unpredictable dynamic environments. DynGMP is built on the key observations surrounding the limitations of existing solutions. As illustrated in Fig. 1, for non-learning-based dynamic motion planners, e.g., Dynamic RRT*, their inherent tree-based exploration process makes the trimming operation incurred by the collision with the changing obstacles very costly. This is because for many visited nodes of the exploration tree, even if they do not directly collide with obstacles, they are still discarded as long as their parent nodes are in the obstacle space. Such an aggressive trimming strategy, by its nature, causes even more expensive regrowth of the exploration tree, a process that already suffers high costs incurred by sequential and extensive sampling and collision checks. On the other hand, if we adopt the state-of-the-art neural planners, e.g., GNN explorer [10], for the unpredictable dynamic environments, it has to repeat the generation of the exploration tree each time when a moving obstacle causes a new collision. According to our performance profiling results measured on modern GPU (Nvidia RTX 3090), the tree exploration (including edge priority sorting and collision checking) takes most of the planning time ($> 70\%$), making such tree regeneration-based dynamic neural planners very expensive.

From the analysis above, it is seen that the high replanning cost, as a fundamental limitation of existing dynamic planners, is mainly caused by the regeneration of the exploration tree. We argue that such expensive re-generation incurred by the aggressive tree trimming and abandonment is inefficient since many discarded nodes contain essential and useful

[1]Department of Electrical and Computer Engineering, Rutgers University.[2]Department of Computer Science, Rutgers University. *Corresponding author. This work is supported by the National Science Foundation (NSF) under Grant CNS-2114220 and IIS-1845888

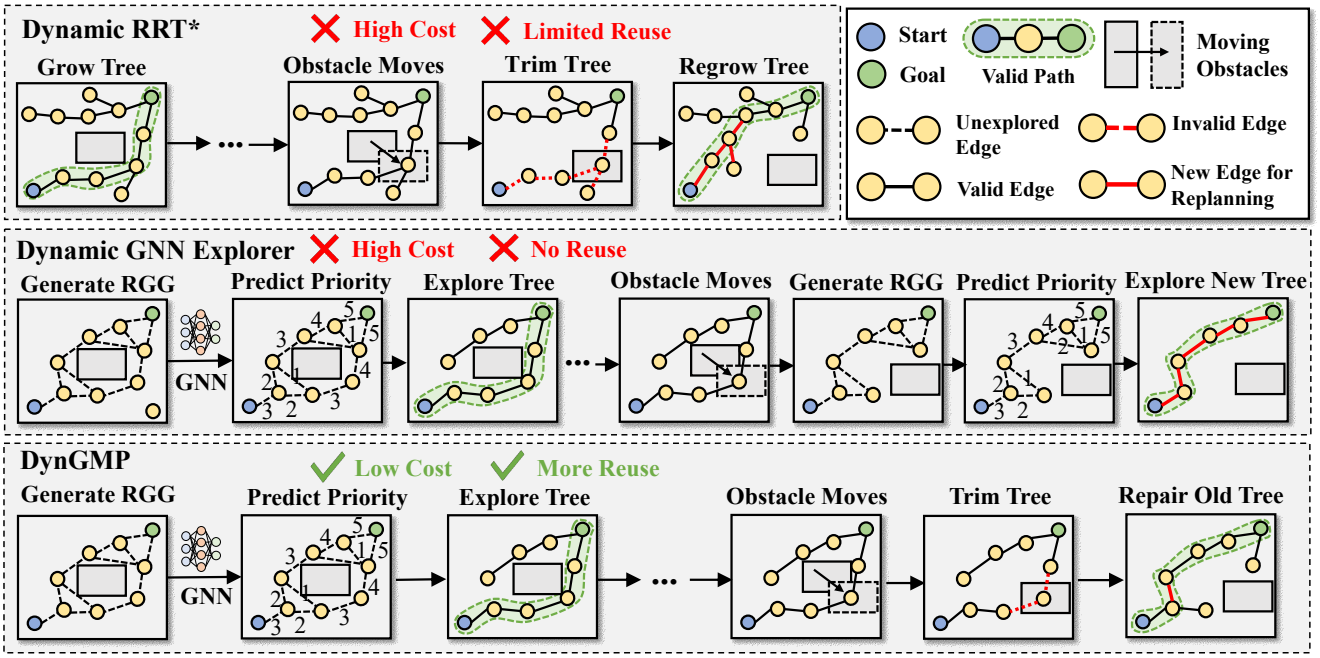

Fig. 1: Different schemes of motion planners when handling dynamic environmental changes. Both the existing non-learning and learning-based (Dynamic RRT* and Dynamic GNN explorer) aggressively invalidate a large portion of the exploration tree; while DynGMP aims to maximally preserve the already explored information to reduce replanning cost.

exploration information, and thus they should be preserved and re-used during the replanning phase. Motivated by the insight, our proposed DynGMP is designed to fully leverage the prior exploration experience to adapt to dynamically changing environments. As shown in Fig. 1, DynGMP aims to maximize the reuse of the exploration tree, bringing low-cost tree repairing and re-growth, simultaneously achieving good replanning performance and efficiency. More details of DynGMP are shown in Fig. 2 and described in Section IV. Overall, the key contributions of this paper are summarized as follows:

- We propose DynGMP, a GNN-based motion planner that adapts to unpredictable dynamic environments. To the best of our knowledge, this is the first neural dynamic planner that can work in challenging scenarios with multiple unpredictably moving obstacles.
- We develop a set of strategies and approaches, including minimization of trimming scope and shortcut smoother, to dramatically improve planning performance while simultaneously reducing planning overhead.
- Empirical evaluations of DynGMP across a broad set of environments confirm that DynGMP can achieve close to 100% success rate with fast planning speed and small path cost. Compared with existing non-learning and learning-based counterparts, DynGMP shows very significant planning performance improvement.

II. Related Work

Dynamic Motion Planning. Dynamic planners aim to find collision-free paths in dynamically changing environments. To achieve this, many existing solutions are based on sampling-based planners (such as RRT/RRT*) that have been successful in static environments. For example, ERRT [1] proposes to adapt to environmental changes by storing waypoints and regrowing search trees. DRRT [3] places the root of the search tree at the goal location to reduce replanning costs by minimizing the number of invalidated branches. Multipartite RRT [12], as a combination of ERRT and DRRT, maintains a set of subtrees that can be pruned and reconnected under the guidance of previous states. Inspired by the superior performance of RRT* over RRT, Dynamic RRT* [2] is designed to perform dynamic planning in working environments containing obstacles with random and unpredictable movement.

Learning-based Motion Planning. Learning-based planners have obtained much attention in recent years. For instance, LSTM is employed to learn to imitate oracle and encode the trajectory history in [13]. [14] integrates known dynamics of robots into the policy neural network and performs model-based reinforcement learning. Following a similar reinforcement learning-based strategy, [9] develops a neural planner with a proper balance between exploration and exploitation. In addition, [15] and [16] propose to learn sampling distributions to guide the search process better, and [8] and [17] utilize different types of neural networks to process structured and unstructured inputs, respectively. Also, considering that a motion planning problem can be interpreted from the perspective of graph search, several GNN-based planners have been reported in recent literature. [18] leverages GNNs to identify critical samples, and [10] adopts a GNN-based solution to reduce the demand for collision checks. However, most existing learning-based planners are designed for static environments. To date, only [11] studies the neural planner in dynamic environments.

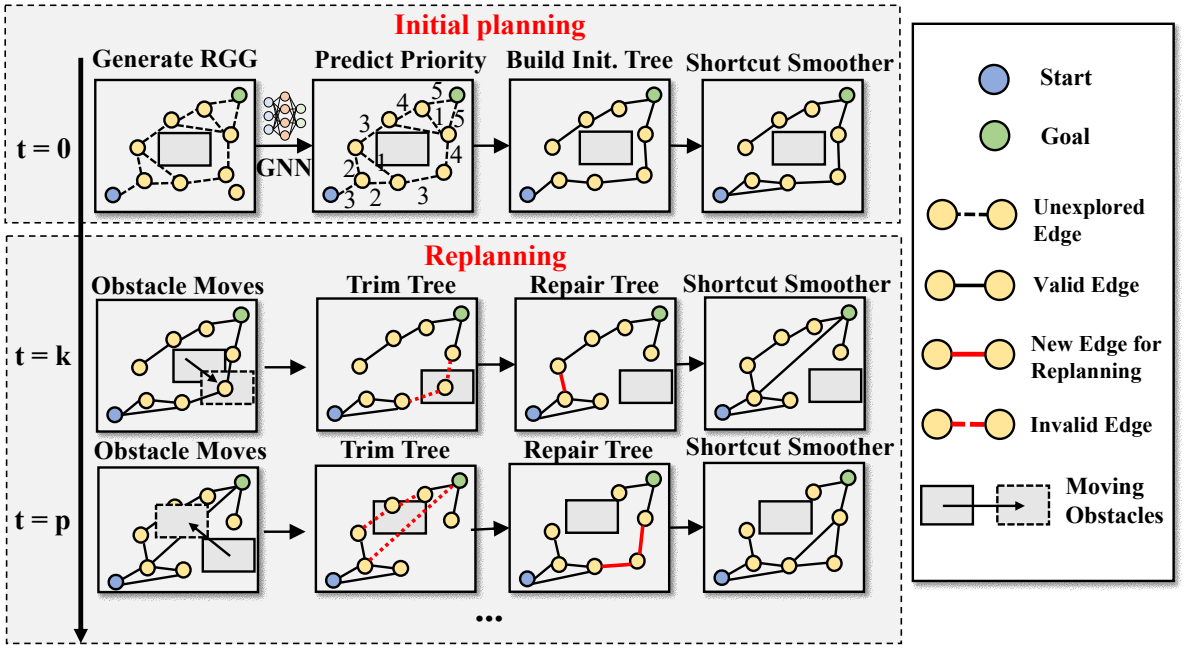

Fig. 2: The overall procedure of DynGMP. In the initial planning phase ($t = 0$), the exploration tree is built on the GNN-generated edge priority. In the replanning phase (each time when environmental change occurs, e.g., $t = k$ and $t = p$), the exploration tree is trimmed and repaired carefully to maximize the original tree structure, lowering replanning cost. A shortcut smoother is always adopted to improve path quality.

However, their proposed GNN-based temporal encoding can only work when the environmental change is predictable, an assumption that does not hold in many real-world scenarios.

III. Preliminaries

Notation. Let $C \in \mathbb{R}^d$ represent the d-dimensional configuration space of the robot. The obstacle and free space are denoted as C_{obs} and $C_{free} = C \setminus C_{obs}$, respectively. Generally, a motion planning problem can be interpreted as the search process over a graph $G = (V, E)$, where V and E are the node and edge sets, respectively. Specifically, each node $v_i \in V \subset C$ represents a configuration state of the robot, and each edge e_{ij} connecting v_i and v_j is a collision-free transition/movement between two states. v_s and v_g represent the robot's start and goal states.

Problem Definition. In practice, obstacles in the environment may have unpredictable dynamic changes. Therefore, obstacles and free space in such scenarios can be parameterized as $C_{obs}(t)$ and $C_{free}(t) = C \setminus C_{obs}(t)$, respectively, where t is time. Dynamic motion planning aims to find a continuous path connecting v_s and v_g, where the robot's configuration at t is always collision-free. More specifically, the planned path in the dynamic environment is denoted as $\pi = \{v_0, v_1, ...v_T\}$, where $v_0 = v_s$, $v_T = v_g$, $e_{i,i+1}(t) \in C_{free}(t)$, $\forall i \in [0, T-1]$.

IV. Method

Overall Framework. Fig. 2 shows the overall framework of the proposed DynGMP. In the initial planning phase, i.e., the state of the robot is v_s at $t = 0$, the random geometric graph (RGG) generated by the random sampling in $C_{free}(0)$, is processed by a GNN-based explorer [10] to predict the edge priority. Then, an exploration tree is built by connecting the collision-free edges with the high priority greedily. A feasible path can be identified upon constructing this exploration tree. To optimize the path quality, the shortcut smoother and GNN smoother [10] can be optionally applied to reduce path cost further. Then, the robot begins its operation following this initially planned path. After the unpredictable random movement of the obstacles at $t = k$, the robot needs to adjust its path to adapt to the dynamic change of the environment in the re-planning phase. To that end, the current exploration tree is first trimmed to remove the nodes and edges involved with collision incurred by the environmental change. Unlike the trim performed in existing works shown in Fig. 1, DynGMP adopts a collision check-free trimming operation (detailed later), maximally preserving the original tree structure and reducing the computational cost. After that, the disconnected exploration tree is repaired and forms a new collision-free path for the current environment, and the path smothers are optionally applied to optimize path quality further. Notice that such operations in the re-planning phase are repeated each time of the environmental change, updating the exploration tree when necessary.

Building Initial Exploration Tree. The exploration tree is initialized at $t = 0$. It is then partially updated each time an environmental change causes a new collision. Following [10], the exploration tree is built on the predicted edge priority in the raw RGG. More specifically, two multilayer perceptrons (MLPs) first embed node and edge into latent space as $x_i^{(0)}$ and $y_{ij}^{(0)}$. Then, a GNN updates the embedding information by aggregating the local information of each node from its neighbors in an iterative way. Denoted RGG by $G = (V, E)$, the update procedure is as follows:

$$
\begin{aligned}
h_{ij} &= \text{Concat}(x_j^{(k)} - x_i^{(k)}, x_j^{(k)}, x_i^{(k)}), \\
x_i^{(k+1)} &= f_g(x_i^{(k)}, \bigoplus_{v_j \in N_{v_i}} (f_v(\text{Concat}(h_{ij}, y_{ij}^{(k)}))), \forall v_i \in V, \\
y_{ij}^{(k+1)} &= \bigoplus (y_{ij}^{(k)}, f_e(h_{ij})), \forall e_{ij} \in E,
\end{aligned} \tag{1}
$$

where k is the iteration index, $\bigoplus$ is an aggregation function, i.e. max operation, Concat is a concatenating function, N_{v_i} represents a set of neighbors of node v_i, and f_v, f_g and f_e are two-layer MLPs. After multiple iterations (e.g., 5), one MLPs can be applied to the updated edge embedding information to achieve exploration priority for every edge in RGG. We leverage edge priority to build an initial exploration tree. Specifically, we initialize the exploration tree with a single state, e.g., the goal state. Then, the initial exploration tree is built by iteratively choosing one edge with the highest priority from the edge subset of RGG that connect with the current exploration tree and adding the edge to the exploration tree if collision-free. This procedure is terminated until the exploration tree reaches the start state.

Trimming Exploration Tree. As shown in Fig. 2, in the event of obstacle movement at $t = k$, the current exploration tree may not be collision-free anymore, requiring the corresponding adjustment for the new environment. To that end, a trimming operation is performed first to remove all the potential collision nodes and edges in $C_{obs}(k)$. A straightforward solution is to perform collision checks on all the edges and nodes and identify the targeted ones, but this strategy brings high computational overhead incurred by extensive collision checks. Instead, DynGMP adopts a simple yet efficient solution to invalidate collision-involved nodes and edges without explicit collision checks.

As shown in Fig. 3, all the nodes within the range of a radius of the moved obstacles and their corresponding edges are removed from the exploration tree. Here the radius is determined by the summation of the robot and obstacle sizes. For instance, for a 7D Snake robot, the robot size is defined as the length of the snake, and the obstacle size is the maximum length of the obstacle's bounding box. By using this method, DynGMP achieves a good balance between maximally preserving the prior exploration record and minimally incurring the computational overhead. Notice that there exists a rare case that two out-of-range nodes are connected via an edge that is in collision with obstacles. To eliminate this risk, we impose a constraint when building the exploration tree – each edge is shorter than the minimum length of the obstacle size. Another benefit brought by this constraint is it also avoids the invalidation of a long collision-free edge due to the small overlap with the obstacles, reducing the demands of the collision check.

Repairing Exploration Tree. Once nodes and edges involved in collisions are trimmed from the exploration tree, DynGMP further performs the repairing operation to make the tree regrow and adapt to a new environment. As shown in Fig. 4, the goal of tree repair is to re-connect the two disjoint subtrees that contain the start and goal nodes without collision as obstacles move. Considering the previously predicted edge priority is prepared for the planning task in the old environment, the edge priority prediction is updated with the latest obstacle information. Upon that, the subtree containing the goal node (v_g) starts to grow by selecting the highest priority edge among all the unexplored nodes connected to that subtree. Notice that sometimes the selected node belongs to another subtree (Case 2 in Fig. 4). In such a scenario, these two subtrees are merged as a new one. Such tree expanding process continues until the growing subtree connects to the subtree containing the starting node (v_s), forming a complete tree. Then, the feasible path for the changed environment can be identified from this repaired exploration tree. Algorithm 1 describes the overall scheme of the tree-repairing procedure.

Algorithm 1: Repairing Exploration Tree

Input : Start $v_s(t)$, Goal v_g, Exploration tree $T(t)$
Output: New path $\pi(t)$, Repaired tree $T(t)$
Data: Raw RGG G, New obstacles O_{new}
Notation: Start subtree $\mathscr{T}_s$, Goal subtree $\mathscr{T}_g$ and Other subtrees $\mathscr{T}_o^{'}$

Derive connected components from $T(t)$ as $\mathscr{T}_s$, $\mathscr{T}_g$ and $\mathscr{T}_o^{'}$;
Predict_priority(G, O_{new});
/* Repair from goal subtree */
Initialize exploration tree T_0 = $\mathscr{T}_g$;
Initialize edge candidate E with edges of G;
$E = E \setminus E_{T_0}$; // exclude explored edges
while *True* **do**
  Select $e = (v, v^{'}), v \in T_0$ from E with highest priority;
  $E = E \setminus \{e\}$;
  if $e \in C_{free}(t)$ **then**
    add edge e to T_0;
    if $v^{'} \in \mathscr{T}_s$ **then**
      /* connect to start subtree */
      $T(t) = T(t) \cup T_0$;
      break;
    if $v^{'} \in \mathscr{G}_o^{'}$ **then**
      /* connect to other subtrees */
      $T(t) = T(t) \cup \mathscr{T}_o^{'}$;
  /* Sample new RGG if edge candidate set is empty */
  if $E == \emptyset$ **then**
    Re-sample and generate new RGG G;
    Predict_priority(G, O_{new});
    $E = E \setminus E_{T_0}$
  if *is_reach_sample_budget()* **then**
    break;
$\pi(t) = Dijkstra(T(t), v_s(t), v_g)$;
return $(\pi(t), T(t))$

Shortcut Smoother. The re-planned path after repairing the exploration tree, though indeed free of collision, may not have high quality because of potential detours. As shown in Fig. 5, it is very common that the path identified from the newly re-connected exploration tree contains unnecessary detours, especially in complicated environments. To address the issue, path smoothing is typically required to reduce detours and improve path quality. However, as shown in Fig. 5(b), after using GNN smoother [10], the customized

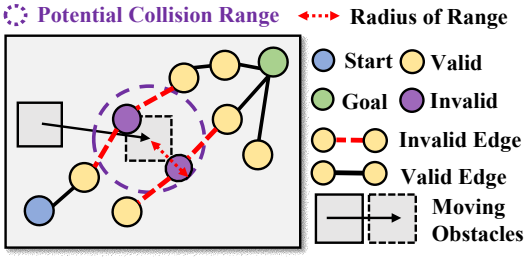

Fig. 3: The scope for trimming exploration tree. All the nodes and the corresponding edges within a radius (maximum length of bounding box of obstacle plus robot) are removed.

path smoother for GNN-based planning, detour issues still exist, causing very limited performance improvement of path quality. We hypothesize one potential reason for this phenomenon is that the existing neural path smoother is designed for planning in the static environment, lacking sufficient generality for unpredictable dynamic changes. To that end, we propose a shortcut smoother to quickly improve path quality at a low cost. As shown in Fig. 5, a smooth window is slid over the found path. Potential shorter edges within the window are examined. Once a shortcut is identified, the smoother adds the shortcut edge to the exploration tree, effectively eliminating unnecessary detours. Notice that following the path diversification methodology [19], [20], DyGMP still preserves the original longer path and the corresponding nodes as a backup to prepare for future dynamic environmental changes. In addition, considering the added shortcut may introduce a long edge, some additional nodes may be interpolated along the edge to satisfy the constraint of the edge length.

Algorithm 2: The Procedure of Shortcut Smoother

Input : Path $\pi(t) = \{(v_i)|i \in [t,T], 0 < t < T\}$ and Exploration tree $T(t)$
Config: Window size w
Output: Smoothed path $\pi(t)$ and Exploration tree $T(t)$

$\pi' = \text{deepcopy}(\pi(t))$;
for $i : 0 \rightarrow (k-w)$ **do**
  for $j : (i+w) \rightarrow (i+2)$ **do**
    if $(v_i, v_j) \in C_{free}(t)$ **then**
      delete $\{v_z | z \in (i,j)\}$ from π';
      add $e_i : (v_i, v_j)$ to $T(t)$;
      break;
return $(\pi', T(t))$

V. Evaluation

A. Dataset and Experimental Setup

We demonstrate the effectiveness of DynGMP on four types of planning tasks: (1) 2D Easy Maze: a 2 DoF robot in 2D map of size 15×15, (2) 2D Hard Maze: a 2 DoF robot in 2D map of size 15×15, (3) 3D Maze: a 3 DoF robot in 2D workspace of size 15×15; and (4) 7D Snake: a 7 DoF snake robot in 3D workspace of size $15 \times 15 \times 1$. Here the difference between Easy Maze and Hard Maze is that the distance between the start and goal points in Hard Maze is much longer than that in Easy Maze, making the planned path much more susceptible to being blocked by the moving obstacles. To simulate the random and unpredictable change, in each working environment four obstacles are set to be able to move independently in eight random directions or remain static. To avoid the potential unsolvability of the planning task incurred by the environmental change, the obstacles are forbidden to move to the goal configuration. In addition, for each environment type, we prepare 2000 planning problem instances to train the GNN model. Each instance contains a different set of random obstacles and a pair of start/goal configurations. The evaluation is performed on another 1000 unseen planning problems with randomly chosen dynamic obstacles. We report evaluation performance by averaging the performance metrics over all the 1000 unseen problems.

B. Baseline

We evaluate the performance of DynGMP and compare it with one non-learning-based dynamic planner (dynamic RRT*[2]) and two learning-based solutions (Dynamic NEXT and Dynamic GNN Explorer). Here because currently there are no reported learning-based planners designed for unpredictable dynamic environments, we modify two neural planners (NEXT [9] and GNN explorer [10]) that are for static environments to adapt for dynamic environments. More specifically, we develop Dynamic NEXT via trimming and re-building the exploration tree of the original NEXT method, and the adopted trimming and regrowing strategy follows the scheme used in Dynamic RRT*, making NEXT capable of dynamic planning. The Dynamic GNN explorer is built on re-executing the original GNN explorer each time when the environment changes. In other words, by interpreting the re-planning task as planning in the new environment with different start configurations and obstacle information, the original GNN explorer can be extended for dynamic planning.

C. Implementation Details

The GNN component of our proposed DynGMP follows the same architecture used in [10]. To be specific, The nodes and edges are encoded into embeddings of 32 dimensions using the separate two-layer MLPs, and the three-loop attention modules encode the obstacle information into the node and edge embedding with the output dimensions of 32 and 32, respectively. The message passing procedure of the GNN aggregates and updates the node and edge embeddings from their local neighbors for five loops, with output dimensions of 32 and 32, respectively. At the output end of the GNN, three-layer MLPs are used to predict the single-value edge priority for each edge. To train GNN, we select ADAM optimizer [21] with a learning rate of 0.001 and batch size of 8. The experiments are conducted on a computer with AMD Ryzen 5600x and GeForce RTX 3090. The dynamic NEXT and Dynamic GNN Explorer are implemented based on the modification of the open source code of [22] and [23], respectively.

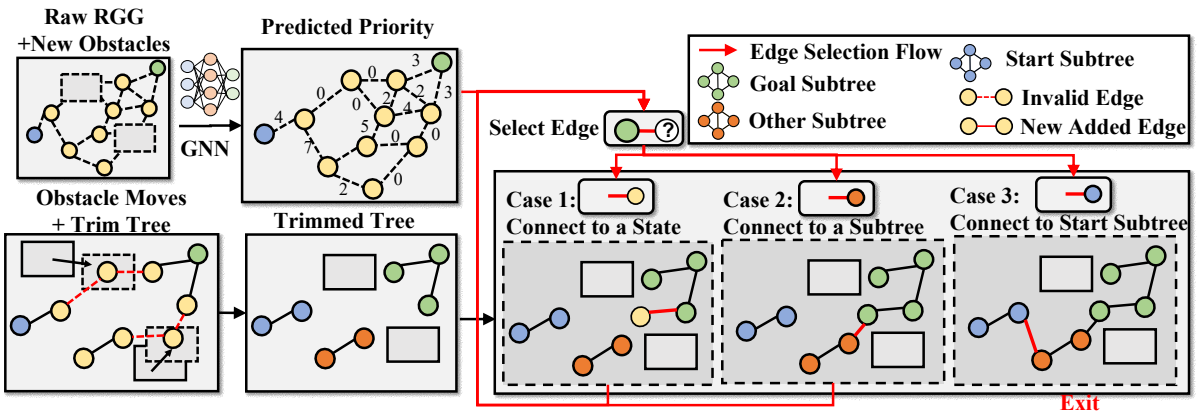

Fig. 4: The procedure of repairing the exploration tree. The re-growth of the tree starts from the subtree containing the goal node (state). Each time this subtree connects to the node associated with a non-explored edge having the highest priority (Case 1). Sometimes such a connection directly brings the merge of two subtrees (Case 2). The growth continues until the start and goal states are connected (Case 3).

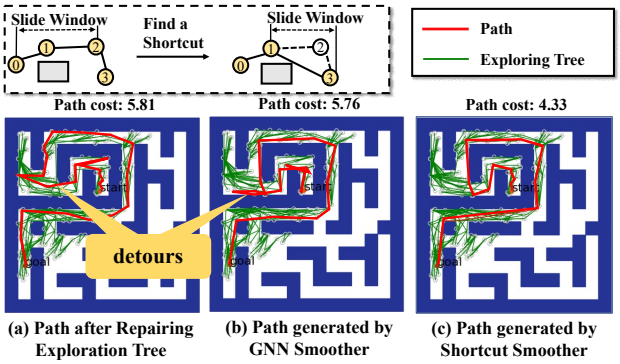

Fig. 5: The sliding window-based Shortcut smoother can reduce the unnecessary detours in the non-smoothed path; while GNN smoother cannot work well for this case.

D. Evaluation Metrics

In our experiments, we evaluate the following performance metrics of different dynamic planners.

1) **Success rate** is the percentage of planning tasks that are solved with a feasible path that connects the start and goal states without collision with the dynamic obstacles.
2) **Collision check** is the number of collision checks performed during the entire planning procedure, including both initial planning and re-planning phases until a feasible path is found or the maximum budget is reached.
3) **Planning time** is the computation time used in the entire planning procedure, including both initial planning and re-planning phases until a feasible path is found or the maximum budget is reached.
4) **Travel distance** is the average of the robot's actual moving distance guided by the dynamic motion planner.
5) **Travel distance with failure penalty** is the adjusted travel distance that penalizes the planners with lower success rates. The reason for evaluating this metric is that the planners are more likely to solve easier tasks with shorter travel distance, but fails in the more difficult tasks with longer distance. For a fair comparison of the travel distance, we introduce a failure penalty which is equal to the multiplication of the maximum robot step size and the number of simulation loops, as the travel distance of the failed tasks.

E. Dynamic Planning Performance

Fig. 6 shows the performance of DynGMP and other baseline dynamic planners. Notice that since GNN explorer is typically concatenated with a GNN smoother [10] for higher path quality, in our evaluation the performance of the Dynamic GNN explorer considers the use of additional GNN smoother. Correspondingly, we evaluate two configurations of DynGMP, as only using its own shortcut smoother and using an additional GNN smoother.

Comparison with Baselines. Overall our proposed solution, no matter with or without GNN smoother, shows very promising performance, guiding the robot to reach the goal on $100\%, 99.9\%, 99\%, 98.9\%$ dynamic planning problems with very low collision check demand and short re-planning time in 2D Easy Maze, 2D Hard Maze, 3D Maze, and 7D Snake environments, respectively. More specifically, as shown in Fig. 6 (a), DynGMP improves the success rate by 3%, 6.5%, 3.2%, and 2% on four environments over the Dynamic GNN explorer. Compared with Dynamic RRT* and Dynamic NEXT, the advantage of our method on success rate is even more significant. Also, Fig. 6 (b) shows that DynGMP requires much fewer number of collision checks in the planning procedure. Compared with Dynamic GNN explorer, DynGMP reduces the demand of collision checks by 72%, 76%, 64%, and 79% in four environments. Compared with dynamic RRT* and Dynamic NEXT, DynGMP reduces collision checks by at least 47%, 61%, 55%, and 66% in different environments. Such a huge reduction further translates to a significant speedup in planning speed. As shown in Fig. 6(c)), DynGMP accelerates the replanning procedure by $2.7\times$, $2.2\times$, $2.4\times$, and $2\times$ in the four environments as compared with dynamic RRT*. Even higher speedups are achieved when comparing DynGMP with Dynamic NEXT and Dynamic GNN explorer. Another benefit brought by DynGMP is the short travel distance. As shown in Fig. 6(d), DynGMP

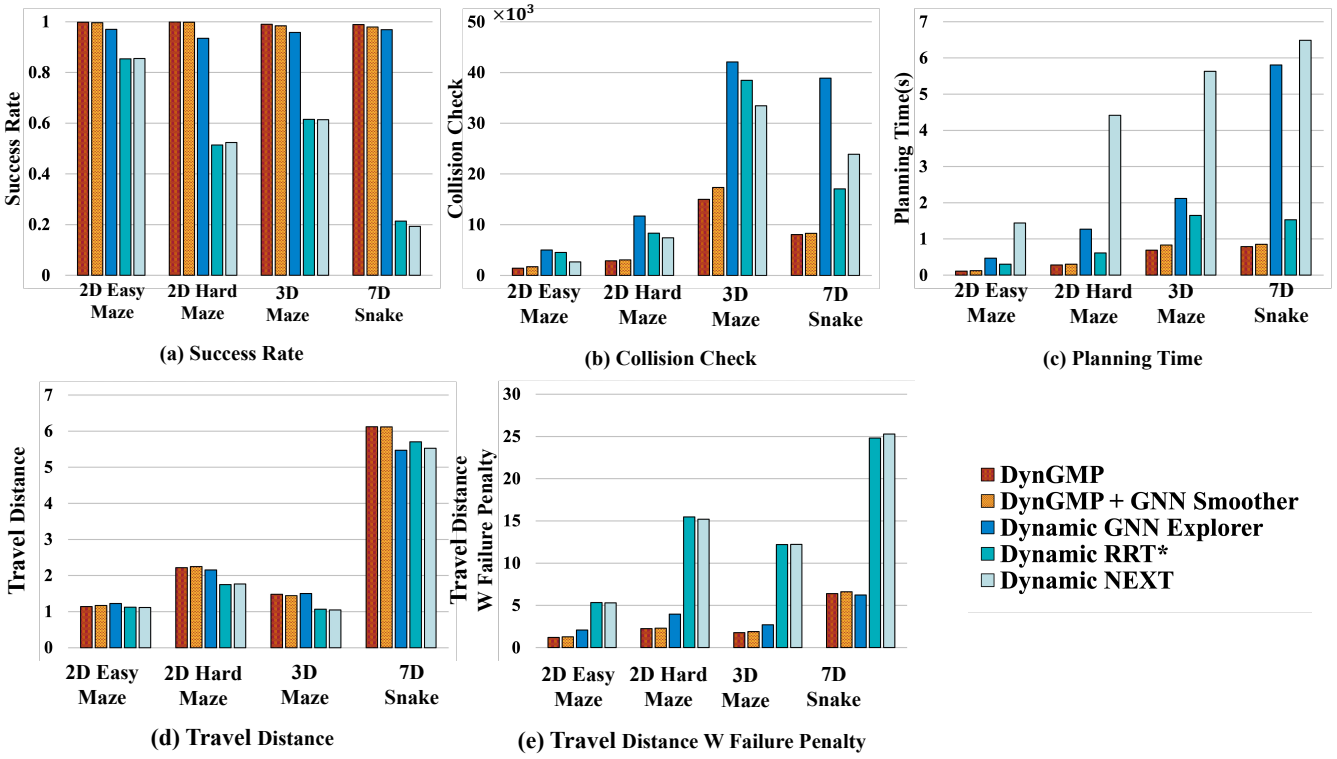

Fig. 6: The planning performance of different dynamic motion planners.

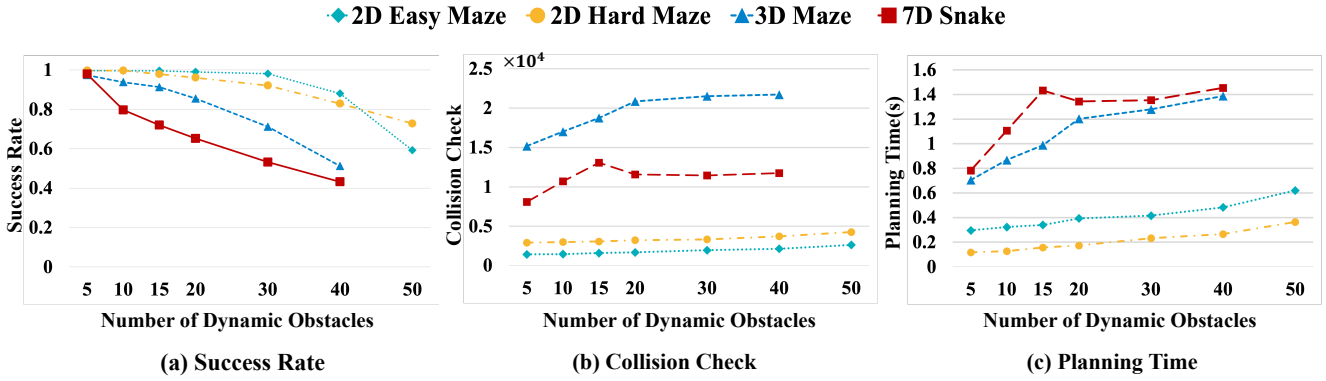

Fig. 7: The planning performance of DynGMP with different numbers of dynamically moved obstacles.

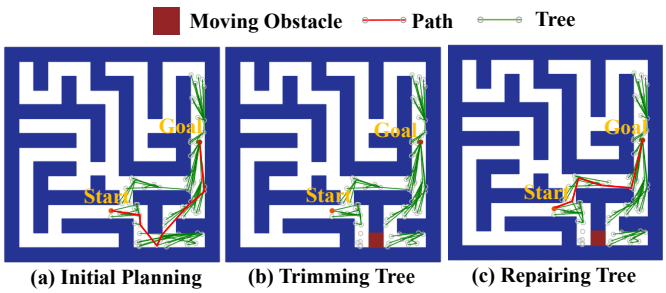

Fig. 8: Visualization of replanning process of DynGMP in 2D Hard Maze environment.

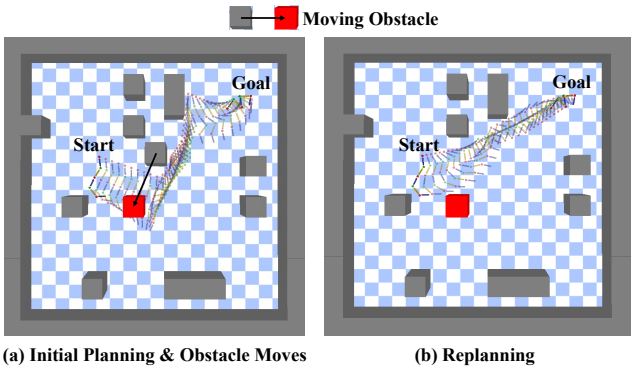

Fig. 9: Visualization of replanning process of DynGMP in 7D Snake environment.

achieves similar travel distances to the baseline dynamic planners. With a much higher success rate, DynGMP shows much shorter travel distance with failure penalty as compared to other non-learning and learning-based dynamic planners (see Fig. 6(e)).

Impact of Path Smoother. Prior works have shown that the proper use of path smoother can improve path quality, especially in a static environment. For instance, GNN smoother is adopted in GNN explorer [10] to reduce path cost. For our proposed DynGMP, the impact of using additional GNN

smoother is very negligible (see Fig. 6). This is because the built-in shortcut smoother in DynGMP can effectively solve the detour problems that GNN smoother does not work well on (see Fig. 5). We also perform an ablation study to compare DynGMP only using shortcut smoother and DynGMP only using GNN smoother. As shown in Table I, adopting the shortcut smoother brings a very significant reduction in the path cost, demonstrating its advantage.

Varying Number of Moving Obstacles. To better understand the capacity of DynGMP, we also evaluate its performance with various numbers of moving obstacles. As shown in Fig. 7, DynGMP can ensure a relatively high success rate for dynamic planning ($> 80\%$) even with 10 obstacle changes in the challenging 7D Snake environment. For 2D environments, DynGMP can achieve above 90% success rate even with more than 30 moving obstacles. Meanwhile, the corresponding time cost is still affordable. For instance, in the scenario of 50 dynamic obstacles in the 7D Snake environment, DynGMP only uses less than 1.5s for the entire planning, demonstrating its fast planning speed.

TABLE I: The impact of using different path smoothers on the travel distance of the robot.

	Without Smoother	**With GNN Smoother**	**With Shortcut Smoother**
2D Easy Maze	2.18	2.15	1.14
2D Hard Maze	2.96	3.01	2.22
3D Maze	2.50	2.36	1.48
7D Snake	6.14	6.05	6.12

F. Visualization

Fig. 8 illustrates the re-planning process of DynGMP in 2D Hard Maze environment. After one dynamic obstacle blocks the current path, DynGMP first trims the tree and enables the exploration tree to be collision-free in Fig. 8(b). Then, DynGMP repairs this exploration tree and finds a new collision-free path (see Fig. 8(c)). The visualization of a similar process in 7D Snake environment is shown in Fig. 9.

VI. Conclusion

This paper proposed DynGMP, a GNN-based motion planning algorithm for unpredictable dynamic environments. Empirical experiments demonstrate that DynGMP achieves close to 100% success rates and delivers competitive path quality with the help of a carefully designed shortcut smoother. Comprehensive experiments suggest DynGMP is a very promising GNN-based dynamic motion planner that can provide at least $2\times$ over the existing solutions.

In future work, we plan to push the ceiling on the number of supported dynamic obstacles higher by designing more intelligent approaches for trimming the exploration tree. When identifying collision nodes and edges, our current design sacrifices the false positive rate to guarantee a 100% true positive rate. This leads to some unnecessary removal of collision-free states and edges. We plan to design a learning-based method to decrease the false positive rate and simultaneously guarantee 100% true positive rate.

References

[1] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *IEEE/RSJ International Conference on Intelligent Robots and System*, IEEE, 2002.

[2] D. Connell and H. M. La, "Dynamic path planning and replanning for mobile robots using RRT," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, oct 2017.

[3] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 1243–1248, 2006.

[4] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998.

[5] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[6] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, pp. 846–894, jun 2011.

[7] M. Otte and E. Frazzoli, "*RRT*^{x}: Asymptotically optimal single-query sampling-based motion planning with quick replanning," *The International Journal of Robotics Research*, vol. 35, pp. 797–822, sep 2015.

[8] L. Lee, E. Parisotto, D. S. Chaplot, E. Xing, and R. Salakhutdinov, "Gated path planning networks," in *Proceedings of the 35th International Conference on Machine Learning* (J. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 2947–2955, PMLR, 10–15 Jul 2018.

[9] B. Chen, B. Dai, Q. Lin, G. Ye, H. Liu, and L. Song, "Learning to plan in high dimensions via neural exploration-exploitation trees," 2019.

[10] C. Yu and S. Gao, "Reducing collision checking for sampling-based motion planning using graph neural networks," in *Advances in Neural Information Processing Systems* (M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds.), vol. 34, pp. 4274–4289, Curran Associates, Inc., 2021.

[11] R. Zhang, C. Yu, J. Chen, C. Fan, and S. Gao, "Learning-based motion planning in dynamic environments using gnns and temporal encoding," 2022.

[12] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, IEEE, apr 2007.

[13] 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)*, pp. 3965–3972, 2019.

[14] T. Jurgenson and A. Tamar, "Harnessing reinforcement learning for neural motion planning," 2019.

[15] B. Ichter and M. Pavone, "Robot motion planning in learned latent spaces," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2407–2414, 2019.

[16] 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)*, pp. 3654–3661, 2018.

[17] X. Zang, M. Yin, L. Huang, J. Yu, S. Zonouz, and B. Yuan, "Robot motion planning as video prediction: A spatio-temporal neural network-based motion planner," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 12492–12499, 2022.

[18] A. Khan, A. Ribeiro, V. Kumar, and A. G. Francis, "Graph neural networks for motion planning," 2020.

[19] M. S. Branicky, R. A. Knepper, and J. J. Kuffner, "Path and trajectory diversity: Theory and algorithms," in *2008 IEEE International Conference on Robotics and Automation*, pp. 1359–1364, IEEE, 2008.

[20] L. H. Erickson and S. M. LaValle, "Survivability: Measuring and ensuring path diversity," in *2009 IEEE International Conference on Robotics and Automation*, pp. 2068–2073, IEEE, 2009.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[22] C. Yu, "gnn-motion-planning." `https://github.com/rainorangelemon/gnn-motion-planning`, 2022.

[23] NeurEXT, "Next-learning-to-plan." `https://github.com/NeurEXT/NEXT-learning-to-plan`, 2019.