Math Foundation of Machine Learning
GNN with Attention Mechanisms for Path Planning

David Felipe Alvear Goyes

December 12, 2023

1 Introduction

Path planning algorithms are a fundamental problem in autonomous systems. Given a defined environment or map where the robotic system navigates, finding a possible path that connects a start with a goal position minimizing a certain heuristic such as distance or energy cost for the robot is considered an optimization problem. Current approaches like RRT or Dijkstra algorithms perform an exploration of the environment sampling states in the environment and divide them into free or obstacle space states. This exploration of the space requires the distinction of valid or feasible states for a robot, in the case of a single differential robot the exploration space is two-dimensional with an orientation consideration (x,y, yaw). The path exploration is done by validating the free space states while connecting a tree of possible states until the start and goal positions are connected in the growth tree. The complexity of this task becomes time and computationally expensive as the dimension of the optimization problem increases. For a manipulator with 7 degrees of freedom or multi-agent systems like 2 robotic arms with 7 degrees of freedom each becomes expensive and time-consuming to validate the possible position that is feasible to connect the possible states tree.

In the following sections is discussed the implementation of a multi-head attention feature extractor with a graph neural network model to perform the path exploration. The path exploration takes the graph generated by the sampled points in the space to then decide the edge to explore in the environment until the destination goal is present in the explored edges and nodes. The designed model was trained using a reference algorithm Dijkstra for path planning and tested in two environments with 2 and 14 degrees of freedom.

2 Methodology

The graph and tree structure of the planning problem makes it possible to use graph neural networks to optimize the sampling and tree expansion in the path exploration pipeline. The study [1] proposes a priority-based approach to select the best suitable nodes to perform the state validation and then perform the node expansion. It proposes a graph neural network that takes the sampled nodes and the graph created using random graph generation RGG with K-NN. The model uses attention mechanisms to learn a representation of the nodes and outputs priority values η where the node with the maximum probability will be the node to be validated and then explored to grow the tree. The sampling and priority computation is performed for several iterations until the final node is found in the tree or the max iteration constraint is reached. The following pipeline describes the path exploration algorithm with priority-based node selection.

2.1 Explorer Algorithm

1. Given a start and goal vertex with the obstacles or map data structure, initialize sampling of nodes from the configuration space $\mathcal{C}$.

   - Sample n nodes from $\mathcal{C}_{\text{free}}$ and add to free space vertices V_f.
   - Sample n nodes from $\mathcal{C}_{\text{obs}}$ and add to free space vertices V_c.

2. Initialize a graph using K-NN creating edges from the different vertices that exist in the possible nodes V and adding start and end nodes $\{v_s, v_g\}$.

$$G = \{V : \{v_s, v_g\} \cup V_f \cup V_c, E : \text{KNN}(V_f) \cup \text{KNN}(V)\}$$

3. Initialize the nodes tree V_τ with the start state and the edges from nodes neighbors using KNN E.

$$E_0 = \text{KNN}(V_f), V_{\tau_0} = \{v_s\}$$

4. Start loop computing the first priorities using the explorer model.

$$\eta = N_E(V, E, O)$$

5. Repeat from $i = 0$ to MAXITERS

   - Select edge to explore using the computed priorities η and E_i
   - Verify that the edge is in the free configuration space, Validation state step

$$e_i : (v_i, v_j) \subseteq \mathcal{C}_{\text{free}}$$

   - Add the node v_j to the exploration tree $V_{\tau_{i+1}}$, the edge to exploration edges $E_{\tau_{i+1}}$ and the free space edges.

$$E_f(\tau_{i+1}) = \{e_j : (v_i, v_j) \in E_{i+1} | v_i \in V_{\tau_{i+1}}, v_j \notin V_{\tau_{i+1}}\}$$

   - Check if Euclidean distance between v_j and goal vertex v_g is less than a threshold. If yes, return the path found π on τ_i.
   - If the created edges in the computed graph is empty, compute new graph and new priorities with explorer model.
     - Sample n nodes from $\mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{obs}}$ creating V_f and V_c
     - $V = \{v_s, v_g\} \cup V_f \cup V_c$
     - $E_i = \text{KNN}(V_f)$
     - $E = \text{KNN}(V_f) \cup \text{KNN}(V_c)$
     - $\eta = N_E(V, E, O)$

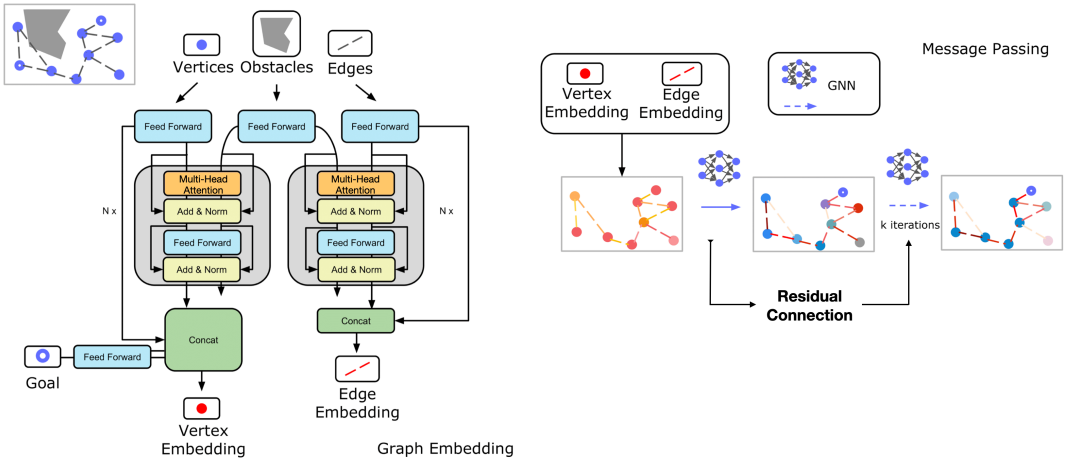

Figure 1: GNN Multi-Head Attention Model

2.2 Explorer Model

The edge exploration priorities are computed using a graph neural network that receives the vertices V, edges E and the obstacles O. The main objective of the model is to maximize the probability of finding a feasible path π by adding edges e_i from the free space to the expanded tree τ_i on each iteration. The nodes features are defined using the vertex position, euclidean distance to the goal and the distance to the goal vertex from each node $\{v_i, (v_i - v_g)^2, v_i - v_g\}$.

$$e_i = \underset{e_k \in E_i \cup E_f(\tau_i)}{\operatorname{argmax}} \; N_E(\eta_k | V, E, O)$$

Figure 1 shows the architecture of the explorer model. The model takes as input the vertices in the free and obstacle configuration space with the created edges in the graph generation. The multi-head attention blocks receive the graph information and create a vertex, edge embedding. The study [1] uses a simplified version of the model using a single attention mechanism and 3 stacked blocks. In this model, 4 heads and 4 stacked transformer blocks with a dropout of 0.3 aim to learn different a more complex patterns in the embedded node features. The vertex and the edge embedding are passed to the graph neural network to update each node's features while taking the information of neighbour nodes. Given that path planning is seen as a graph process see can see that the neighbor nodes and the goal nodes are important to detect the priority of edge exploration. Figure 1 shows the message passing improving the iterative process using a residual connection between steps in the graph updates. Finally, The output of the model is an adjacency matrix with the entries of the probability of exploring each edge.

2.3 Training Process

The explorer model is trained using a supervised approach. The training starts with the problem definition using the obstacles, start vertex, goal vertex, and the generated graph. The model is designed and trained to imitate a path-planning algorithm to find the shortest path in the graph. [1] uses the Dijkstra algorithm as the reference algorithm to create a path and guide the training. The deep learning model will receive the problem statement and then compute the priority of exploration for the next edge following the same strategy as Dijkstra. The model explores the generated graph until there are no edges on it creating an optimal path until the destination node is present. For training, the same loss function is defined by [1] where the true distribution for the next edge to explore is always one and the log of the predicted probability will give the direction to update the model.

Parameter	Value
Maximum Sampling Number	1000
GNN Batch Size	100
Training Epochs	40
Learning Rate	1×10^{-3}
Batch Size	8
Optimizer	Adam
Regularization	Dropout (0.3) and L1
Training Scenes	48

Table 1: Training Parameters

$$L_{\eta_E} = -log\gamma_{N_0}, \gamma_k = \frac{e^{\eta_k}}{\sum_{j \in E_i \cup E_f} e^{\eta_j}}$$

The hyper-parameters and training definition are listed in Table 1. The model is trained generating a determined number of training scenes to perform the exploration and compute the loss function for the selected scenes. For every eight processed problems, the model weights are updated. The loss function uses the L1 regularization norm to encourage sparse solutions.

3 Experiments and Results

The environments used for training the model are Maze 2D see figure 2 and 2KUKA robots with 7 degrees of freedom 3. The model was trained with the defined parameters in table 1 and the performance is compared with RRT and Dijkstra algorithm for path planning. The use of dropout and L1 regularization achieved a good performance in the validation step and avoided overfitting resulting in stable learning.

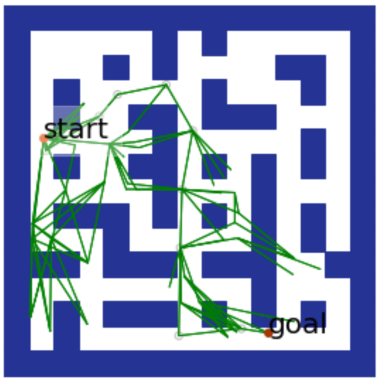

Figure 2: Maze 2D

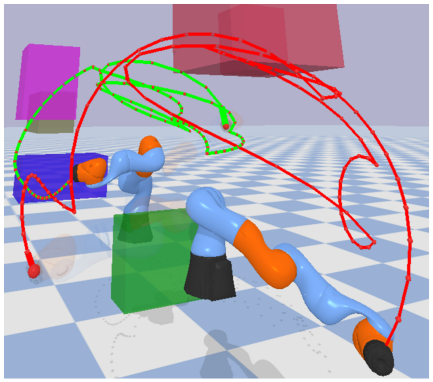

Figure 3: KUKA 14 DoF

3.1 Maze 2D Results

For the evaluation of the simplest case with 2 degrees of freedom we have a a better performance in time complexity using the exploration model with GNN and attention mechanisms. Dijkstra algorithm was used as an oracle to train the model to select the best edge to explore on the graph. Given that the edge selection is done using the parallelization capabilities of GPUs we can evidence the improvement on the time necessary to compute the paths. Figure 4 and 7 show the reduction in time for path planning in a 2D environment. The most expensive algorithm in time is RRT, next is Dijkstra and the cheapest is the proposed model.

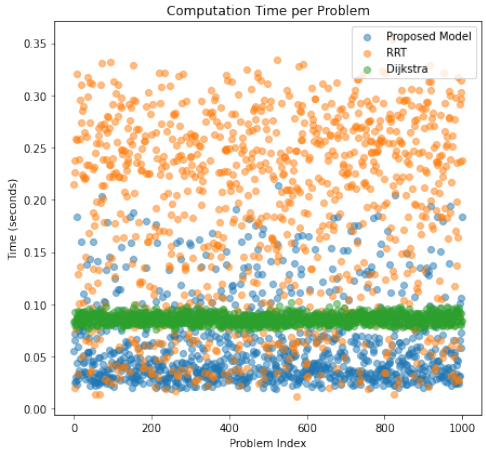

Figure 4: Maze 2D

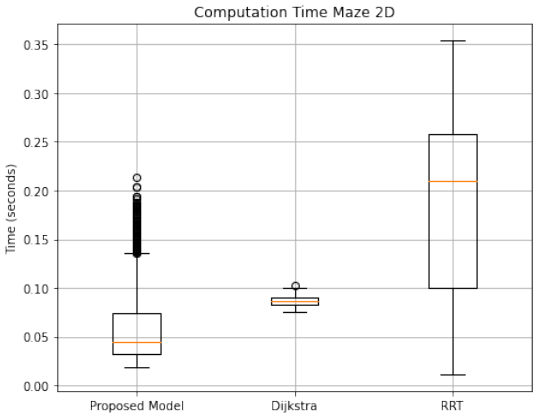

Figure 5: BoxPlot Maze 2D performance

3.2 KUKA 14 DoF Results

The evaluation of the model in the most complex case of 14 degrees of freedom is shown in figure 6 and 7. We can see in 6 that using the deep learning model (blue dots) we have less than 1/4 of the time used with the dijkstra algorithm, and also it improved the time needed by RRT to find a path. Figure 7 shows a better estimate of the performance of RRT, Dijkstra and the attention model to solve 1000 different path planning problems. The improvement in time between RRT and Dijkstra is clear but the use of Attention mechanisms reduced the time of collision checking using the prioritized-based edge exploring. One reason that Dijkstra is taking a lot of time is in the graph computation and then the iteration over the nodes and edges to find the path. The use of attention mechanisms and GNN with GPUs make the process of edge evaluation parallelizable, and it is one of the reasons for reducing the time in finding a path.

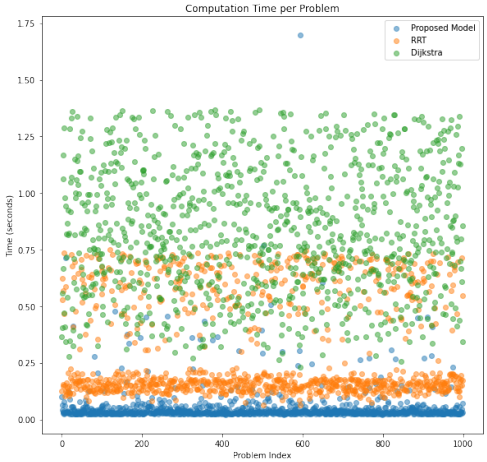

Figure 6: Exploration in 1000 different problems

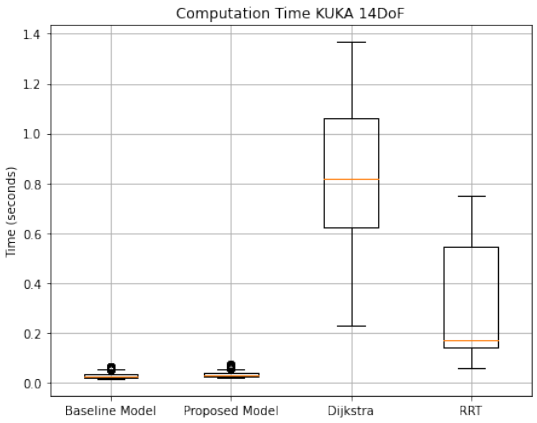

Figure 7: BoxPlot KUKA performance

4 Conclusions

There was implemented a model with multi-head attention mechanisms and a graph neural network with residual connection in the message passing to imitate an oracle to explore a graph and find a path that connects a start and goal positions. The oracle used was the Dijkstra algorithm and the explorer model followed the reasoning of the oracle to find a path in the

generated graph. The exploration algorithm creates a graph using sampled points from the environment and expanding a free space tree with selected edges computed by the exploration model, the reduction in time lies in the re-computation of a new graph until there are no edges in the previous graph or the destination is already in the explored tree.

The training of the deep learning model followed the suggested approach in [1] with modifications in the hyper-parameters, maintaining the loss defined in the study. The use of regularization techniques such as dropout and L1 were implemented to stabilize and generate convergence in the loss. The training scenes were incremented in the training concerning the original paper to reduce the noise gradient computation and the L1 regularization stabilized the weights updates. The model was trained for a simple case in a 2 dimension environment and the most complex case in a multi-robot application in 14 degrees of freedom.

In the evaluation of the model, we can see that the reduction in time is considerable in the simplest case. However, the reduction in time in 14 DoF is more due to the time complexity of traditional algorithms like Dijkstra and RRT for large dimensional environments. We can conclude that one of the key effects in reducing the time in path planning is the use of the parallel computation capability of GPUs to evaluate the best edge to explore and reduce the time spent in the iterative process of traditional path planning algorithms. Moreover, the use of the proposed model with multi-head attention mechanisms and GNN is important in complex environments, the reduction in time in simple 2-dimensional environments is still cheaper using Dijkstra and RRT in terms that they can be computed using CPU and still have reasonable time performance.

References

[1] C. Yu and S. Gao, "Reducing collision checking for sampling-based motion planning using graph neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 4274–4289, 2021.