Deep Reinforcement Learning and Enhanced Pathfinding

ECE5984 Deep Reinforcement Learning

Timo Thans {timot@vt.edu}, **Sophia Rubsamen**
Department of Electrical and Computer Engineering, Virginia Tech

***Abstract*—Graph problems and their corresponding algorithms have been pivotal in the development of computing, impacting numerous fields including robotic routing. This paper demonstrates that learning methods can effectively solve shortest path problems in robotic routing systems and details the design of the environment. Additionally, we explore how embeddings in the Graph Neural Networks (GNN) are crucial for developing a model for an optimal policy that generalizes to larger and more complex environments. Our work contributes to RL4CO, a unified codebase with state-of-the art software, by expanding the variety of environments and methods available. The project's code is available at https://github.com/TimoThans33/ece5984_drl_ai4spp.**

I. Introduction

A. Graph Combinatorial Optimization

Graph combinatorial problems have been studied by computer scientists for decades as these problems impact many fields, incudling fields such as robotic routing in an autonomous warehouse.

These problems typically involve finding the best arrangement or trajectory, minimizing or maximizing a given objective metric such as minimizing the number of visited nodes in a robotic routing problem. The traditional graph algorithms, while effective in static graphs, often fall short in environments where the graph's topology and edge weights can change in real-time due to obstacles, varying conditions, or updated priorities.

B. Robotic Domains

In the robotic routing problem, graph combinatorial optimization plays a critical role in a robotic delivery system navigating a grid-based envrionment. These systems objective could be, to efficiently deliver items to specified locations on a grid using multiple agents, while considering obstacles, other robots, energy and time constraints. Current implementations use famous algorithms such Dijkstra's or A* to solve the environment and find the shortest path between the current position and the the target delivery location.

Deep Reinforcement Learning (DRL) offers a promising approach to further enhance the efficiency and adaptability of robotic routing problem. DRL techniques have proven to be succesfull in many more complex envrionments, but even in simpler environments such as the grid-based graph in a robotic delivery system could benefit from these state-of-the art algorithms. The robotic routing problem could very well be formulated as an agent tasked with navigating a grid and delivering items with certain rewards. Such an agent could make actions based on the edges in the graph (i.e, adjacent grid nodes) and make observations (i.e., target location, current location, obstacles, other robots) to maximize cumulative rewards. We hypothesize that deep neural networks are particularly interesting in these types of problem, because of their ability to approximate complex policies and generalize well to changing conditions and potentially revolutionize logistics and automation.

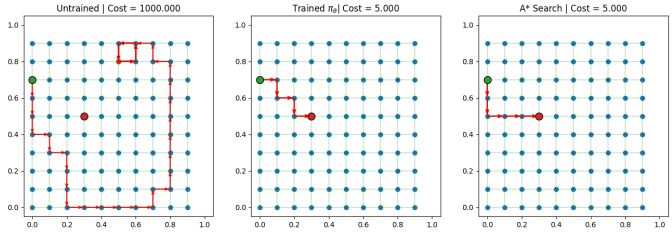

Fig. 1: Pathfinding performance across different policies in a 100-node grid environment. From left to right: Untrained policy, Trained policy, and A* search, with paths shown in red. The untrained policy results in a high cost (1000.000). In contrast, both the trained policy and the A* search achieve a cost of 5.000, demonstrating the effectiveness of our trained policy not only in surpassing the untrained baseline but also in matching the performance of the traditional A* search, underscoring the significance of our contributions in this paper.

In this paper, our biggest contribution is to utilize 2 DRL methods to solve a new environment, the shortest pathfinding problem. Previous methods evaluated other more researched environments such as the Travelers Sales Person (TSP) and the Capacitated Vehicle Routing Problem (CVRP). We chose to research the attention model, and the Policy Optimization Multiple Optima (POMO) models and do a comparative analysis to the performance of a pathfinding A* search algorithm in this new shortes pathfinding problem. Both DRL algorithms have proven to be succesfull in different envrionments and have been well established in the graph deep learning research community making them the obvious choice for our envrionment. We use the Attention model exactly as show in [2] with an auto regressive policy as well as the POMO model shown in [3] also with an auto regressive policy. By making minor changes to the embeddings of the models, as is common for GNNs, we show that these models successfully converge to an optimal solution in our grid-world with target nodes.

II. Methods

A. Shortest Path Problem Environments

1) Static: For building our environment, we heavily rely on the work done in the rl4co project [1] developed and maintain by the AI4CO research group at KAIST. This software package, allowed us to build on top of their default TSP envrionment, a well-known combinatorial optimization problem that involves finding the optimal path to visit a list of cities exactly once and return to the starting city.

Our grid-based environment works slightly different, by allowing the agent to find the shortest path from a randomized start and end node and constraining the action space by only visiting neighbouring nodes in the graph. The graph is maintained in an adjency matrix represented by a 2D tensor to make optimal use of the vectorization performance improvements available in our grid environment. Our environment allows for batch processing and can be used to train with GPU. The reward is designed as the negative amount of steps taken in the environment once it reaches the target node, or when the amount of steps are maxed out at 100 the reward is set to -1000 which highlights the penalty to the agent for not finding a solution and preempt the episode for faster training.

2) Dynamic: To highlight the differences between the DRL methods and traditional algorithms, we also developed a dynamic environment. At a certain probability in this environment obstacles are added by invalidating certain edges between nodes in the adjacency matrix. This challenges the agent by having to find the optimal path around the obstacles that it was not trained on.

B. A star search

The A* search algorithm is the most popular pathfinding solution because of its effectiveness across a range of applications, including the robotic routing problem. In this problem, the heuristic can be designed to minimize the distance to a target node by following the least nodes as possible, which will prioritize the exploration of trajectories that offer the lowest estimated cost. The key benefit of A* is that it allows for exploration and exploitation of the heuristic, enabling it to find an optimal solutions efficiently even in very large and complex environments.

Although A* search is widely used in many route planning scenarios, its effectivenss is limited in implementations with strict real-time constraints. To speed up compuation, many implementations store the results and perform a look-up in a data structure upon receiving a request, instead of solving each instance from scratch. This approach, however, does not fully capitalize on the benefits of A* search.

C. Attention Model

Attention Models have been successfully used to enhance deep learning models by focusing on the most important input elements, by dynamically adjusting the weights of different parts of the input.

The Attention Model proposed in [2] implements a graph attention network, integrating a graph structure through mask operations. The encoder generates embeddings for all nodes in the graph, these could include information but not limited to x and y coordinates of all nodes. The context embeddings capture information regarding the existing node embeddings and more importantly task-specific details, such as current node, end node, and other environment information.

The input nodes are processed through three sequential layers, each with an eight head attention mechanism, followed by feed-forward sub-layers. The diagram in 2 and 3 shows the implementation of the attention model.

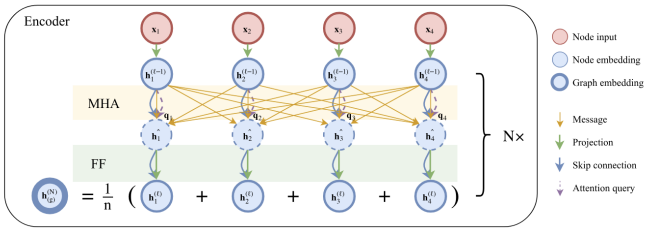

Fig. 2: Attention based encoder for the TSP problem. [2]

On each tour, the decoder takes the graph embedding and node embeddings (x, y and distance to target). At each step, the context embeddings in the decoder, take the current node and the target node. Nodes that cannot be visited because they are not it's neighbours are being masked.

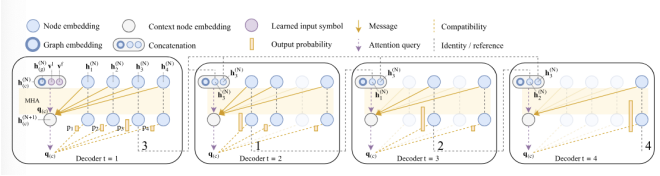

Fig. 3: Attention based decoder for the TSP problem. [2]

From the decoder, a policy could sample form the resulting probability distributions. The model is trained by performing gradient descent using REINFORCE on a loss function, which is defined as the expectation of the tour length. In [2] a *deterministic greedy rollout* baseline policy is proposed as opposed to a more common actor-critic algorithm.

D. Policy Optimization Multiple Optima

Symmetries in RL methods for solving CO problems allow for aggressive optimizations. These symmetries are leveraged by the POMO method, by performing parallel roll-outs and instance augmentation.

We employed the POMO model as suggested in [3], which uses the same underlying architecture as the attention model using the encoder and decoder with the attention layers.

During training, POMO explores from multiple starting nodes allowing the agent to solve the same problem repeatedly from many different angles, exposing it to a variety that would otherwise be unused.

To combine the parallel tours, a shared baseline is used, which performs gradient updates in the exact same manner as

Algorithm 1 POMO Training

procedure TRAINING(training set S, number of starting nodes per sample N, number of training steps T, batch size B)
  initialize policy network parameter θ
  for $step = 1, \ldots, T$ **do**
    $s_i \leftarrow \text{SAMPLEINPUT}(S) \quad \forall i \in \{1, \ldots, B\}$
    $\{\alpha_i^1, \alpha_i^2, \ldots, \alpha_i^N\} \leftarrow \text{SELECTSTARTNODES}(s_i) \quad \forall i \in \{1, \ldots, B\}$
    $\boldsymbol{\tau}_i^j \leftarrow \text{SAMPLEROLLOUT}(\alpha_i^j, s_i, \pi_\theta) \quad \forall i \in \{1, \ldots, B\}, \forall j \in \{1, \ldots, N\}$
    $b_i \leftarrow \frac{1}{N} \sum_{j=1}^{N} R(\boldsymbol{\tau}_i^j) \quad \forall i \in \{1, \ldots, B\}$
    $\nabla_\theta J(\theta) \leftarrow \frac{1}{BN} \sum_{i=1}^{B} \sum_{j=1}^{N} (R(\boldsymbol{\tau}_i^j) - b_i) \nabla_\theta \log p_\theta(\boldsymbol{\tau}_i^j)$
    $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
  end for
end procedure

the attention model except that the loss function is summed over the parallel exposures. However, there are different baselines that can be used with the POMO method.

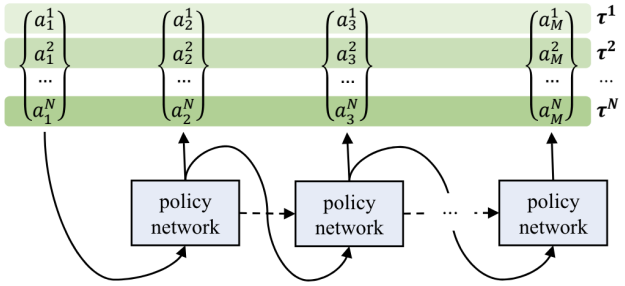

Fig. 4: POMO method for multiple trajectory generation in parallel with a different starting node.

Algorithm 2 POMO Inference

procedure INFERENCE(input s, policy π_θ, number of starting nodes N, number of transforms K)
  $\{s_1, s_2, \ldots, s_K\} \leftarrow \text{AUGMENT}(s)$
  $\{\alpha_k^1, \alpha_k^2, \ldots, \alpha_k^N\} \leftarrow \text{SELECTSTARTNODES}(s_k) \quad \forall k \in \{1, \ldots, K\}$
  $\boldsymbol{\tau}_k^j \leftarrow \text{GREEDYROLLOUT}(\alpha_k^j, s, \pi_\theta) \quad \forall j \in \{1, \ldots, N\}, \forall k \in \{1, \ldots, K\}$
  $k_{\max}, j_{\max} \leftarrow \text{argmax}_{k,j} R(\boldsymbol{\tau}_k^j)$
  return $\boldsymbol{\tau}_{k_{\max}}^{j_{\max}}$
end procedure

At the inferences, instance augmentation is performed additionally to generate additional start and target nodes which allows for exploration of more diverse trajectories towards the optimal solution.

In our experiments, we generated 8 additional augmentations for the input to represent different trajectories. At the end, the start nodes are again sampled and multiple greedy rollouts are performed and the optimal trajectory is chose.

III. Experimental Results

The testing phase does not require an environment similar to the one used during training. In our experiments, we used a grid environment with size of 100, where the start and target nodes were randomly selected for each episode. During the inference step, the policy methodically navigates the environment, executing greedy rollouts based on the available actions. We evaluated all trained policies against both an untrained baseline and an A-star search algorithm to verify their validity. In the visual renderings of the episodes, the environment is made clear by displaying actions in red, start and target nodes in green and red respectively, and edges from the adjacency matrix of the graph in green.

Immediately after only 2-3 epochs, showing the results in Fig. 1, the trained policy is able to exploit the environment and find a policy better than the untrained policy. And, in many cases also better than the A-star search trajectory.

A. Static Grid Pathfinding

The primary objective in designing the static environment was to validate our hypothesis that deep learning methods can effectively solve the shortest path problems.

Method	Mean	Median	Deviation
A-Star	7.038	7.00	3.496
Attention Model	6.730	6.00	3.40
POMO	6.623	7.00	3.277

TABLE I: Pathfinding results of different models in static environment

We selected the Attention model and the POMO model for initial trials, and both demonstrated their efficacy by consistently finding correct solutions across the entire test set. Notably, the trained policies often opted for more aggressive trajectories towards the target, despite identical rewards as ilustrated in Fig. 5. We hypothesize that this behavior results from the graph's embedding within the model, allowing the agent during training to exploit this information and find paths that seemingly increase the probability of achieving a high reward through zigzagging movements. This insight underscores a significant advantage of deep learning methods: even when multiple solutions yield the same heuristic or reward, there is often a single optimal path, and our trained models are adept at identifying it.

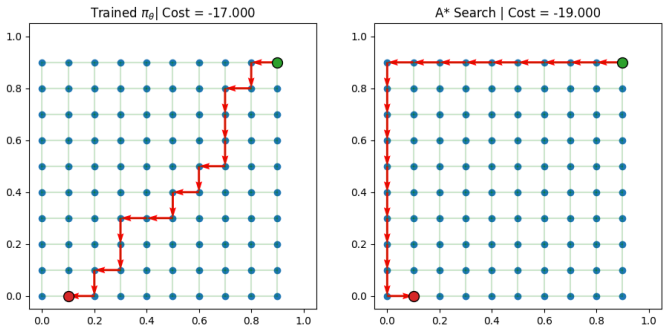

Fig. 5: Comparison of pathfinding strategies: left is a DRL model's path in a static grid with a cost of -17, showing its superior performance over the A* search with a cost of -19

B. Dynamic Grid Pathfinding

The dynamic environment is designed to demonstrate the generalizability of the trained policies, specifically by using the policy that was initially trained in a static environment. This approach further solidifies the argument for the DRL method's broad applicability, offering a deeper comparison of the inherent limitations of the A-star algorithm.

Method	Mean	Median	Deviation
A-Star	7.489	7.00	3.760
Attention Model	6.948	7.00	3.504
POMO	7.052	7.00	3.438

TABLE II: Pathfinding results of different models in dynamic environment

Due to a fundamental difference in rollout of the two methods, DRL exhibits a clear advantage. Unlike the A-star algorithm, which must solve the graph from scratch each time, the DRL method can efficiently apply the same trained model across various scenarios.

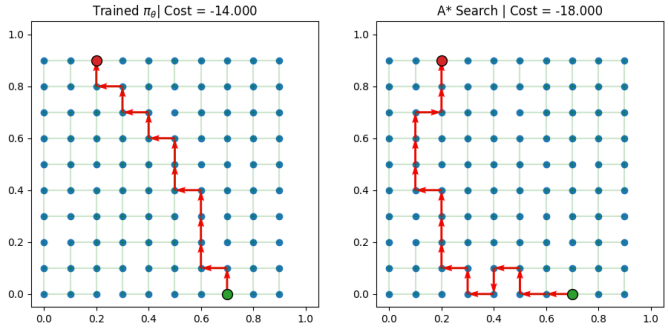

Fig. 6: Comparison of pathfinding strategies: left is a DRL model's path with a cost of -14, showing good generalizability in dynamic environments. Right shows an A* search path with a cost of -18.

In Fig.6 an example roll-out is shown, in which the DRL agent structurally is able to find more optimal solutions even with the A-star search being allowed to solve the environment from scratch every time the obstacles changes, which in real-time scenarios is usually not feasible.

IV. Discussion

In certain scenarios within our randomized dynamic environment, approximately once in every 500 runs, the agent failed to reach the target node. As illustrated in Fig. 7, these failures occurred whenever the agent was required to execute non-greedy (i.e, sacrificial moves that initially move away from the target) to ultimately achieve the final reward. This specific issue presents a compelling direction for future research.

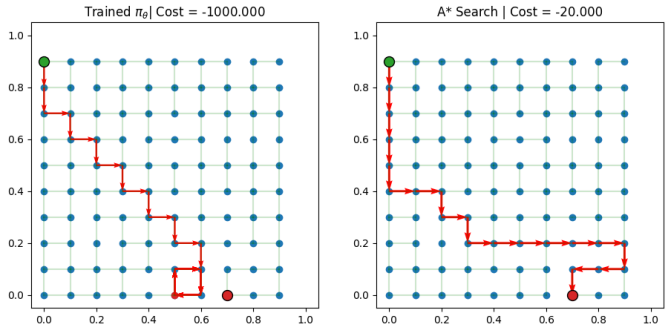

Fig. 7: Comparison of pathfinding strategies: left is a DRL model's path with a cost of -1000, showing a highly greedy and suboptimal policy. Right shows an A* Search path with a cost of -20, highlighting its guaranteed to find a solution.

A. Graph Neural Neworks: Node Embedding

A GNN can be used to learn the representation of nodes in a graph, these are known as node embeddings. In the literature, where mostly TSP problems are studied, it is generally acceptable to embed only the coordinates of the nodes in the graph. However, as we show in this paper, the embeddings can be used to find more complex relationships between nodes than just coordinates.

Another accaptable direction would be, to investigate how the edges and obstacles can efficiently be embedded in the embedding. In real applications it is common that the agent has good knowledge of the obstacles in the environment.

V. Contributions

The primary contribution of this research is the integration of a shortest path-finding problem into the framework of existing deep reinforcement learning methods designed for combinatorial challenges. This study underscores the significant distinctions between heuristic graph algorithms and their deep learning equivalents, demonstrating that neural networks are not only viable but also promising options for advancing solutions to graph-related problems, such as robotic routing.

Additionally, we explore the potential of graph neural networks and the utility of their embeddings in tackling complex challenges. As an undergraduate student at Imperial College London and while being a Robotic & AI Scientist in industry, I have worked on motion planning software for multi-robot systems and they have always caught my imagination, I am very interested in investigating how my newly acquired knowledge in DRL can be used to enhance the capabilities of my existing applications.

In this project, my primary contribution was to the coding aspect. Using my experience in PyTorch and Python, we were able to adapt and integrate the attention model, adding necessary functionalities to address our specific needs such as batch processing and vectorization for efficient GPU processing. Furthermore, I designed and executed the experiments on my home deep learning machine.

This summer, I plan to develop a software package designed to solve more complex shortest path problems using DRL, which will be beneficial not only to my team at Prime Vision, but also potentially to the broader open-source community, including merging the package within rl4co.

References

[1] Federico Berto, Chuanbo Hua, Junyoung Park, Minsu Kim, Hyeonah Kim, Jiwoo Son, Haeyeon Kim, Joungho Kim, and Jinkyoo Park. RL4CO: a unified reinforcement learning for combinatorial optimization library. In *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*, 2023. https://github.com/ai4co/rl4co.

[2] Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, February 2019.

[3] Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Seungjai Min, and Youngjune Gwon. Pomo: Policy optimization with multiple optima for reinforcement learning. *arXiv preprint arXiv:2107.05646*, July 2021.