Applying Reinforcement Learning to the Shortest Path Problem

Dillon Wilson

Abstract

The Shortest Path Problem is extremely well known and has a variety of applications in the real world. Although an efficient and robust algorithm, A*, exists to solve this problem it is still the subject of plenty of published papers attempting to apply a variety of new methods to solve it. In this paper, a recent modification to the Transformer-based neural network, the Decision Transformer, will be applied to a large, weighted graph in an attempt to evaluate it's performance. This model will be compared a variety of well known Reinforcement Learning algorithms to gauge it's performance.

Introduction

The shortest path problem is one has existed for quite some time, and has had a variety of algorithms applied to it over it's lifetime. Many likely are aware of the A* algorithm, which is one of the most famous algorithms so solve the problem due to it's ability to handle negative edge weights, unlike Dijkstra's algorithm, and it also considers the distance already travelled in the path. Although this problem is technically solved, it is still an area of constant research due to it's many applications. Examples of the shortest path problem in the real world range anywhere from Navigation Apps like Google Maps, network optimization, transportation and logistic optimization to name a few. So if a faster version of A*, or an alternative algorithm altogether, can solve the problem more efficiently all of those applications can make use of the upgraded algorithm.

Background

A cornerstone of this project is the usage of a very new type of model in the field of Reinforcement Learning known as the Decision Transformer. This model is a modification of the same transformer model that has been made famous due to it's application in the field of Natural Language Processing. It can be adapted with minimal changes for the goal of training a Reinforcement Learning agent. As with language transformers, Decision Transformers treat their input data sequentially meaning the agent will consider previous actions based on an attention mechanism presented in research that first introduced the concept (Vaswani et al. 2023). At each time step, a tuple of reward, action, and state is fed into an embedding layer that creates a representation of that data. Next, a positional encoder marks at which time step that tuple was processed by the model. The embedding is then fed into the transformer model, producing an output that is taken into a decoding layer. At the end of this process, an action is predicted that should have the highest likelihood of the best possible future reward. An illustration of this is provided at the end of this section in Figure 1.

In order to to truly gauge the performance of this advanced model, a plethora of very well known algorithms were implemented along with it. In the research done by Chen et al. 2021, they compare their decision transformer to multiple Q-learning variations in the well known Atari and Gym environments, however they neglected to replicate these experiments for the shortest path problem. In this paper, those experiments will be performed and extended to far larger graphs. Each of the implemented models will be discussed in further detail in their own subsection in the Methodology section.

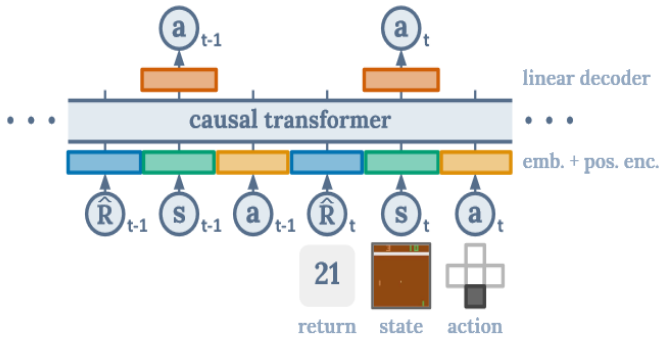

Figure 1: Decision Transformer (Chen et al. 2021)

Related Work

A crucial project that will be referenced constantly while working on this project is the research that introduced the concept of the Decision Transformer by adapting a language model to the problem of Reinforcement Learning (Chen et al. 2021). Luckily enough, there is even an experiment contained in that paper that is directly related to finding the shortest path in a graph. Due to the nature of the problem

being essentially a graph where the edges are interstates and the vertices are distribution centers, research from 2023 around the application of Decision Transformers for graphs may prove useful. It is applied to a different type of problem, primarily Atari games, but will likely include useful details that will be applicable to this project (Hu et al. 2023). Another article that may prove very useful when attempting to formulate the problem correctly is known as the Meal Delivery Problem. It has it's differences, but retains the main elements of a source, a carrier that moves an item between locations, and a destination where the item needs to arrive in reasonable times. This article details an approach to solve this problem using deep Q-learning networks (Jahanshahi et al. 2022). Finally, in research done proposing delivery techniques that do not require human contact, an encoder-decoder based architecture is applied to the problem of Reinforcement Learning (Wu et al. 2023). This research may prove useful as a second source of implementation details, albeit using a different model architecture.

Environment

For this project, a plethora of python packages have been found and utilized in order to create a robust and easy-to-use platform for performing experiments. At the core, there are three crucial packages; OpenAI's Gym, NetworkX, and RouteGym.

OpenAI Gym is extremely well known and likely needs no introduction for those familiar with Reinforcement Learning research. It is and end-to-end solution to handle the training process for an Reinforcement Learning agent. It allows the user to create an environment for their problem, simulate actions, and receive the results of those actions like the agent's new state, the reward received, and whether or not the new state is terminal. On top of handling these various functions of the Reinforcement Learning process, gym is one of the most commonly used packages in the field, and as a result a large portion of the existing code and research relies on it.

The second package is NetworkX, this library is used to create custom, weighted graphs. For this project, that means graphs can be implemented that parallel the US Interstate System with custom weights equal to the distance between various nodes. There are also a handful of useful functions for creating visualizations of the graph objects, but they tend to become unmanageable with a large number of nodes.

The final package is RouteGym, this is a third-party library created for Gym that allows the user bring their own NetworkX graph and create a custom environment. This environment is capable of recognizing which nodes are connected to each other, their respective weights, and can even visualize an agent progressing through the graph if desired. The combination of these three packages facilitate the creation of an environment that is a capable testing ground for all of the various models that will be tested for this project. Along with the major three, a handful of packages are used as-needed including PyTorch, Transformers, Pandas, etc.

Using these packages, two graphs were created that would be used to test the various Reinforcement Learning algorithms presented in this paper. The full graph consists of 48 nodes and 78 edges, with edge weights equal to the real world distances between those nodes using the US Interstate system. There is also a smaller subset graph that is used to test prototype algorithm designs. This allows for faster tests due to the smaller size, and a more explainable result due to the far smaller size.

Action-space For this project, the action-space for the various agents consists of the set of nodes in the graph itself. This means that in the event that an agent chooses action 0, it is attempting to move from it's current state to node 0 in the graph. However an additional consideration must be made when selecting actions, that being that none of the graphs used in this project are fully connected. So an agent cannot be allowed to transition between two nodes unless there is an actual edge in the graph that connects those nodes. For the simpler Q-learning algorithms tested, that is a trivial hurdle, simply disallow the agent from selected an action that is not valid based on the current state. This limits action selection to only valid choices, and results in a Q-table that is populated for the valid actions in each state. But for the more advanced algorithms like Deep Q-Learning and Decision Transformers, a more sophisticated approach must be taken. This problem will be explored in depth in the implementation sections for the Deep Q-Learning and Decision Transformer models.

State-space The state-space, much like the action-space, at it's simplest is the set of nodes in the graph. Where each element of that set has a corresponding state. The state an agent is in can simply be represented as a single value which is sufficient for the simple models tested in this project. However for the deep learning approaches, the state is represented as a one-hot vector where the 1 indicates the agent's current state. As an experiment to be conducted in the future, the state could be expanded to include additional elements like state adjacency in order to allow the agent's to make more informed decisions as the cost of a larger state-space and likely longer training time required.

Reward For this stage of the project, the reward structure has been set up in a very simple way to meet the desired outcome of the agent. Simply put, the reward an agent receives for taking an action is equivalent to the opposite of that edge weight. So if the agent chooses an action that takes it from node TX to node MS and that edge has a weight of 402, the agent will receive a reward of -402 for that action. This implementation was chosen because the desired behavior of the agent is not to minimize the number of actions required to traverse the shortest path, but rather the real distance. Although this design does work quite well for the Q-learning algorithms, it possibly could be a culprit to the instability exhibited by the Deep Q and Decision Transformer models as will be seen in the Result section. Due to this, it will potentially be necessary to tailor a reward structure to the particular model it is being used for, but that will be explored at a later stage of this project.

Implementation

This section will detail the specific details around how the various agent's were implemented and altered to fit the dynamics of this particular problems, it is split into subsections for the various categories.

Q-Learning and Double Q-Learning As one of the most widely known and understood Reinforcement Learning algorithms, Q-Learning serves as a sort of control in this project. It is well known where the algorithms weaknesses lie, however it is very effective and unsurprisingly produced great results during this phase of the project. At it's core, the Q-Learning algorithm relies on the following equation in order to inform it's policy.

$$Q(s,a) = (1-\alpha)*Q(s,a)+\alpha*(R+\gamma MaxQ(s',a)) \quad (1)$$

Where R is the immediate reward the agent receives after taking action a in state s, alpha or learning rate is a parameter between zero and one, and gamma or discount factor is also a parameter between zero and one. Alpha determines the scale in which the existing q value should be modified by the second half of the equation, known as the temporal difference target. Gamma is a parameter that discounts the estimated value of the future state. The only difference between Q-learning and Double Q-Learning is that while Q-Learning uses one estimator in the form of a Q-table, Double Q-Learning uses two. This solves the major issue present with the Q-Learning algorithm, that being the habit of the policy to overestimate Q values. Double Q-Learning uses a slightly altered update equation to accommodate this change.

$$Q_a(s,a) = Q_a(s,a)+\alpha*(R+\gamma MaxQ_b(s',a)-Q_a(s,a)) \quad (2)$$

When an update occurs, the target Q-table is selected randomly. The equation about shows the update specifically for the Q(a) table. The only modification to these algorithms is to accommodate the restrictions that must be placed on the agent as described in the Action-space section.

Deep Q-Learning Deep Q-Learning brings a modification to the standard approach of Q-learning with respect to how the Q-values for a particular action-state pair are determined. Where a table of values is used in Q-learning to keep track of these values, Deep Q-Learning uses a neural network to estimate these values. The current state is inputted into the network and a set of values of the same dimension as the action-space is outputted. The maximum of these values would be the action selected by the agent, however as mentioned in the previous section, there is a restriction placed on the action-space. So, the valid action with the highest value as determined by the neural network is selected. The nature in which the network updates it's weight is done through a process known as 'experience replay'. As the agent performs actions during an episode, those transitions are recorded into a buffer and upon reaching a determined size, the agent samples these transitions and updates the weights of it's network. However, if only one network was used by the agent it would be attempting to estimate Q-values that are being adjusted every time the network updates. For this reason, two networks are utilized. One network updates after each step assuming the buffer is the appropriate length, and the other network, known as a Target Network, is updated occasionally using the current values in the first network.

$$Q - Target : R + \gamma max_a Q(S', a) \quad (3)$$

$$Q - Loss : R + \gamma max_a Q(s', a) - Q(s, a) \quad (4)$$

Where Q-Target is used to predict the action the agent should choose and Q-Loss is the value used to perform backpropagation and update the network.

Decision Transformer The implementation of the Decision Transformer model in the project currently is likely the most simple out of all three. As it currently stands, the Decision Transformer operates by storing the transitions of the last twenty time-steps and encoding them into a token. This token can then be input into the Transformer and an action will be outputted. Unlike the other algorithms, the Decision Transformer is trained by using a dataset of expert trajectories that help inform the model what decisions should be made. This implementation is based on a design by the creator of the algorithm and was applied to the Hopper gym environment, so it is not quite tailored to this specific problem yet. A large amount of work in the second half of the project will be focused on building a more adapted Decision Transformer.

Evaluation

Based on the desired outcome of the agent, the metric for evaluation will be very two-fold. Because training efficiency is an important aspect of this project, tracking cumulative reward by episode gives a good idea of how quickly the agent's policy is converging on the optimal solution to a problem. It also provides useful insight into how stable the agent's learning is, high oscillations between episodes likely means the agent's policy is flawed or that the agent is exhibiting too much exploratory behavior. However, if the algorithm is working correctly it is likely that a chart will show erratic behavior early in it's training but it should stabilize around a reward that is equivalent to the length of the shortest path.

The second metric to be tracked for each agent is it's ability to find the true shortest path in a graph. This can simply be represented as a percentage of successful trials versus total trials. The agent's solution for shortest path can easily be verified by using a built-in function in NetworkX that uses A* to calculate the path between two nodes and it's length. It is expected that, when working correctly, all of the algorithms will be capable of finding the shortest path through a graph. Due to this, the true test will be how efficiently the agent learns in the window of 500 episodes, as well as it's rate of finding the shortest path especially when it's policy is imperfect. The Deep Q-Learning algorithm makes use of two algorithms.

Results

Full Graph Currently, the results of this project show promising progress on a working implementation of Q-Learning and Double Q-Learning. There are two plots included in Appendix that show how two agents perform on a particular test using these two agent designs. These two agents were tested on the full graph described in the Environment section, and in every trial they were able to find the shortest path through the graph.

Subset Graph

The Deep Q-Learning and Decision Transformer models are currently in a prototype phase and being tested on the subset graph while they are incomplete. Both models seems to exhibit quite a bit of instability and clearly are not showing an effective policy. Charts showing these models performance on their own trial are also included in the Appendix. Interestingly, the results from the Decision Transformer model clearly show that the agent has found some sort of maximum in terms of the reward, but the policy can not consistently replicate this result.

Future Goals

As the current worst performing model presented in this paper, Deep Q-Learning needs the most work out of all the algorithms implemented so far. There are multiple possible causes that could be leading to the instability and poor performance of the model currently. It is possible that due to exclusively using negative rewards, the neural network is not being properly updated and therefore an optimal policy not found. This could be tested by simply altering the reward structure to include some aspect of positive rewards. Alternatively, the necessary restriction placed on the action space could be result in the network not updating correctly. Currently, if an invalid action is selected by the network, a transition is stored in the buffer with a large negative reward. This is done to inform the network that selecting an invalid action is the wrong choice. This rather simplistic approach seems like it should work based on how the overall algorithm functions, but a more elegant solution likely exists. Based on research done in 2022 on Deep Q-Learning networks, it was found that handling invalid actions through negative rewards is an inferior solution compared to invalid action masking (Huang and Ontañón 2022). Their work also states that invalid action masking scales to larger environments better than negative rewards, making it a more viable solution for larger graphs.

The Decision Transformer also needs improvement as demonstrated by it's results. The likely culprit for it's poor performance is most likely due to training though. The model presented in this paper was given only 500 episodes of trajectory data to train on. This was done to keep a level playing field as all algorithms were given the same number of episodes to train. Although a fair approach, it is likely not the most optimal for a transformer-based model which tend to need large amounts of training data. However, as an alternative, a second approach exists. In 2022, a paper was published with the sole topic of applying Decision Transformers to the shortest path problem using an adapted GPT-2 model (Memisevic, Panchal, and Lee 2022). This could serve as a potential backup if the larger dataset does not deliver the desired results.

Although the Q-Learning models performed very well in their tests, they are quite old. As an additional challenge, a more modern version of the Q-Learning algorithm could be implemented. One option, presented for comparison in Chen et al. 2021, is the Conservative Q-Learning model (Kumar et al. 2020). And a second option would be the Self-Correcting Q-Learning model (Zhu and Rigotti 2021). Both of these attempt to solve the underlying issue in Q-Learning outlined earlier, it's problem of overestimating Q-values.

Finally, a simple addition to this project could simply be the addition of larger graphs. Currently the largest sits at 48 nodes, but that number could be expanded to 100 by adding additional nodes in between the current ones. Ideally, the current largest graph would serve as a 'small' graph, a graph of around 100 nodes would serve as a 'medium', and a final graph of roughly 125-140 nodes would serve as a 'large'.

Timeline

This sections outlines the remaining tasks of this project along with estimations of when those milestones should be reached.

Timeline	Estimated Completion
Improvement of Deep Q-Learning Model	3-25-24
Refine Decision Transformer	4-8-24
Test Models on larger graphs	4-22-24
Refine Q-Learning Models	4-26-24
Finalize Models and conduct final tests	5-3-24

Conclusion

This paper outlines a project proposal centered around the idea of creating multiple Reinforcement Learning agents with the purpose of solving the shortest path problem on a graph that represents the United States Interstate System. Currently there are 4 algorithms implemented; Q-Learning, Double Q-Learning, Deep Q-Learning, and the Decision Transformer. At this point, the Q-Learning models are showing very good results of the full sized graph, while the Deep Q and Decision Transformer models show lackluster performance and need work to reach an optimal state.

References

Chen, L.; Lu, K.; Rajeswaran, A.; Lee, K.; Grover, A.; Laskin, M.; Abbeel, P.; Srinivas, A.; and Mordatch, I. 2021. Decision Transformer: Reinforcement Learning via Sequence Modeling. arXiv:2106.01345.

Hu, S.; Shen, L.; Zhang, Y.; and Tao, D. 2023. Graph Decision Transformer. arXiv:2303.03747.

Huang, S.; and Ontañón, S. 2022. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. *The International FLAIRS Conference Proceedings*, 35.

Jahanshahi, H.; Bozanta, A.; Cevik, M.; Kavuk, E. M.; Tosun, A.; Sonuc, S. B.; Kosucu, B.; and Başar, A. 2022. A deep reinforcement learning approach for the meal delivery problem. *Knowledge-Based Systems*, 243: 108489.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative Q-Learning for Offline Reinforcement Learning. arXiv:2006.04779.

Memisevic, R.; Panchal, S.; and Lee, M. 2022. Decision Making as Language Generation. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2023. Attention Is All You Need. arXiv:1706.03762.

Wu, G.; Fan, M.; Shi, J.; and Feng, Y. 2023. Reinforcement Learning Based Truck-and-Drone Coordinated Delivery. *IEEE Transactions on Artificial Intelligence*, 4(4): 754–763.

Zhu, R.; and Rigotti, M. 2021. Self-correcting Q-Learning. arXiv:2012.01100.

Appendix

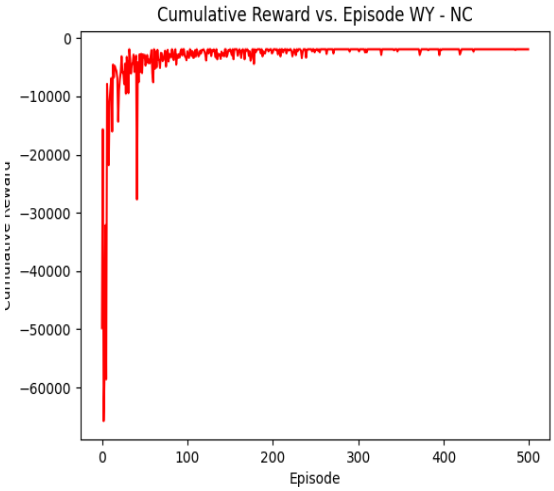

Figure 2: Q-Learning - Full Graph

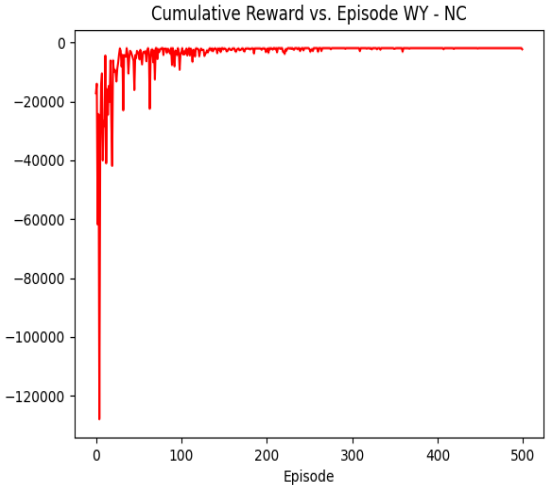

Figure 3: Double Q-Learning - Full Graph

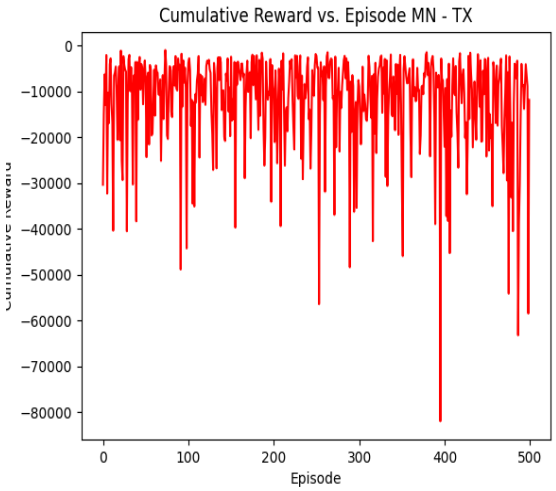

Figure 4: Deep Q-Learning - Subset Graph

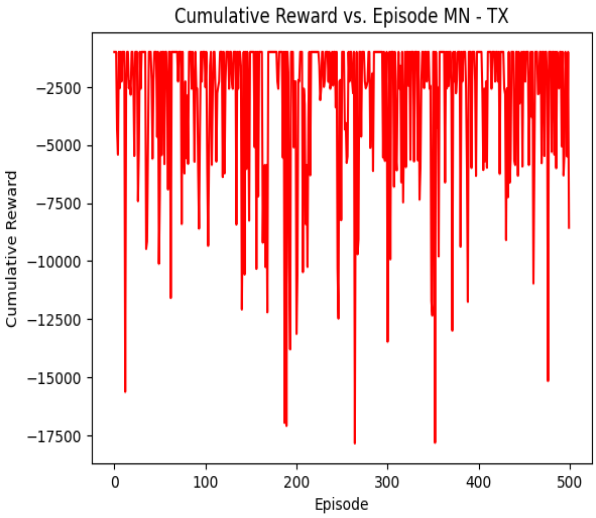

Figure 5: Decision Transformer - Subset Graph