Stitching with Decision Transformers

Sam Liang
Advisors: Shunyu Yao and Karthik Narasimhan
Princeton University
saml@princeton.edu

Abstract

The Decision Transformer is an architecture that models reinforcement learning as a sequential modeling problem and attempts to serve as a replacement for traditional offline reinforcement learning methods. Decision Transformers match the state-of-the-art performance on Atari and OpenAI Gym benchmarks. It also shows an ability to stitch together different segments or parts of training data to generate novel sequences that achieve the desired return. In this paper, I investigate the Decision Transformer's ability to stitch by training it on segmented graph data and seeing whether, at test time, it can stitch together seen segments to generate the shortest path in the graph.

Contents

1 Introduction

The Decision Transformer [2] is a novel architecture that models reinforcement learning as a sequential modeling problem. By doing so, it can take advantage of the many advances in natural language processing [2], specifically the transformer architecture [15] which is ubiquitous in natural language processing and has also been adopted in computer vision. Furthermore, it can benefit from the extensive research in stable training of transformers and better utilize large datasets while being more generalizable. It can also perform credit assignment via self-attention [2] without any extra modifications.

To further investigate the generalizability of the Decision Transformer, I expand on their brief discussion of using the Decision Transformer to find the shortest path in a graph. Given random walk data, the Decision Transformer successfully generated the shortest path when prompted to do so with 15.8% of generated shortest paths being novel [2]. In this paper, I test if the Decision Transformer can truly stitch together segments that achieve the desired return by isolating it in environments where it must stitch to achieve the desired return.

2 Related Work

Stitching. No subsequent work has investigated whether Decision Transformers can proficiently and consistently stitch together segments of data to form desired trajectories. However, the idea of offline reinforcement learning algorithms applying stitching has been discussed by Fu et al. [3]. They discuss scenarios where the dataset is generated from an agent that is solving a different but related task to the one the offline reinforcement learning agent is solving and stitching together the training data can help solve the task at hand. This allows the offline reinforcement learning algorithms to generalize better. Stitching is also prevalent in computer vision in which the goal is for an algorithm or model to put together images with overlapping regions to produce a panorama. The idea is the same in that we want the model to learn to stitch together components to form the desired product. Recent work has focused on how to improve this [9] [8].

Connecting Skills via Offline RL for Generalization. Singh et al. [13] introduces an approach called COG which trains the agent with an extra dataset D_{prior} along with the normal dataset D_T. D_{prior} contains trajectories obtained from agent(s) solving other unrelated tasks, and D_T contains trajectories generated from an agent solving the task at hand. The agent can learn different skills from

D_{prior} which allows it to successfully complete the task when starting from a wide range of initial conditions that it may not have seen in D_T by stitching together the learned behaviors in D_{prior} to those learned from D_T.

GPE + GPI for Policy Stitching. Barreto et al. [1] explores a similar idea of enabling an agent to learn a new task faster by using its skills obtained from previous tasks. They combine generalized policy evaluation (GPE) and generalized policy improvement (GPI) to perform policy stitching which is the agent possibly following a different policy at every state in order to achieve more reward over following any one policy.

Behavior Cloning. In Behavior Cloning [11], a classifier is trained to replicate the expert's policy from a dataset of state-action pairs generated by an expert operating in the environment. The classifier maps states to actions which is used to generate the action to take in a state.

Upside Down Reinforcement Learning. Similar to Behavior Cloning, Upside Down Reinforcement Learning [12] transforms reinforcement learning into a form of supervised learning where the model learns to map rewards to actions. In Upside Down Reinforcement Learning, the model also takes as input a desired return like the Decision Transformer and outputs actions.

Trajectory Transformer. The Trajectory Transformer [5] shares even more parallels with the Decision Transformer. It also models reinforcement learning as a sequential modeling problem, but uses the transformer architecture to capture the distribution of trajectories and beam search as the planning algorithm.

Conservative Q-Learning. The state-of-the-art traditional offline reinforcement learning algorithm is Conservative Q-Learning (CQL) [6] which improves on Q-Learning by learning a lower bound on the Q-function to inhibit an agent's tendency to overestimate. This alleviates the distributional shift problem [7] which is when an agent overestimates the value of taking an action they rarely saw in the training dataset at test time.

3 Background

3.1 Reinforcement Learning

Reinforcement learning refers to an agent learning what actions to take so that they maximize their reward in a static or dynamic environment that sends them feedback after each action [14]. The agent usually has some goal in this environ-

ment.

3.1.1 Markov Decision Process and Trajectories

Reinforcement learning can be modeled as a Markov Decision Process (MDP). The MDP defines a set of states S; a set of initial states S_0; a set of actions A available to the agent; a transition function $T : S \times A \times S \to [0, 1]$ that defines the probability of going to state i when taking action a in state j; and a reward function $R : S \times A \times S \to \mathbb{R}$ that defines the reward of taking an action in state i and arriving in state j. The agent traverses through the MDP by taking actions in each state and arriving in another state until it reaches the goal state. A trajectory τ is a sequence of returns-to-go, states, and actions defined in the following [2]:

$$\tau = \left(\hat{R}_0, s_0, a_0, \hat{R}_1, s_1, a_1, \ldots, \hat{R}_n, s_n, a_n\right)$$

starting from an initial state s_0 and specifying chronologically which states the agent visited and what actions they took in those states. Returns-to-go are mathematically defined as [2]:

$$\hat{R}_t = \sum_{i=t}^{n} r_i$$

where r_i is the reward received at timestep i and n is the last timestep. In other words, returns-to-go is the sum of future reward from the current timestep.

3.1.2 Offline Reinforcement Learning

Reinforcement learning is divided into online and offline reinforcement learning [7]. The Decision Transformer operates in offline reinforcement learning [2]. In offline reinforcement learning, the agent cannot interact with the environment nor receive new information about the states and rewards from the environment. Instead, the agent has access to a trajectories dataset. The trajectories dataset is a set of trajectories that were generated by another agent solving the relevant task in the environment. The goal of offline reinforcement learning remains the same: to learn how to maximize reward in the environment except from trajectory data obtained from a different agent and not being able to interact with the environment to obtain feedback.

3.2 Sequential Modeling Problem

In a sequential modeling problem, the input comes in as a sequence with some time or positional component to it. Furthermore, the input at some time t depends on all the prior inputs. For example, predicting the next word in a sentence is a sequential modeling problem as each word has an order and the next word depends on the context or the previous words. The Decision Transformer projects reinforcement learning into this domain by setting up the reinforcement learning task as predicting the next action to take based on the history of past returns-to-go, states visited, and actions taken [2].

3.3 Transformers

The Decision Transformer uses the GPT [10] architecture which is based off of the transformer [15]. The transformer consists of stacked encoder and decoder layers where the output of the last encoder layer is fed into every decoder layer. Each encoder layer contains a multi-head self attention sublayer and a feed forward sublayer. The multi-head self attention layer is the key innovation of the transformer. It contains multiple self attention layers. Each self attention layer acts as a way for the transformer to learn which parts of the sentence to pay attention to with respect to the current token. By having multiple of them, each self attention layer can learn to focus on separate things in the sequence. For the ith token, it's respective output in the self-attention layer z_i is:

$$z_i = \sum_{j=1}^{n} \texttt{softmax}(\{\langle q_i, k_{j'} \rangle\}_{j'=1}^{n})_j \cdot v_j.$$

Figure 1: Self Attention Calculation [2]

Each decoder layer contains a multi-head self attention sublayer, a feed forward sublayer, and a masked multi-head self attention sublayer. The masked multi-head self attention sublayer is the same as the multi-head self attention sublayer, except it masks out the future tokens so that the model can perform autoregressive prediction. For a more detailed explanation, refer to the transformer paper [15].

3.4 Shortest Path Directed Graph Problem

The shortest path directed graph problem is finding the shortest path between two vertices on a directed graph, where the shortest path is defined as the path whose sum of edge weights is the smallest. In Figure 2, the shortest path is Start→2→4→Goal because its path weight is 4 which is lower than all other path weights.

3.5 Stitching

In this paper, stitching is the ability for an algorithm or model to piece together different segments or components of training data and produce an original solution (in the sense that this solution is not in the training data) that achieves the desired return in the task. For example, in Figure 2, 1→3, 1→3→4, or Start→1→3→4→Goal are segments or components. If the training dataset only consisted of segments of length 1 (e.g. 1→3) from Figure 2, then stitching is piecing together Start→2, 2→4, and 4→Goal to produce Start→2→4→Goal, when prompted to find the shortest path and never seeing Start→2→4→Goal in the training dataset. It is this property that we investigate in the Decision Transformer.

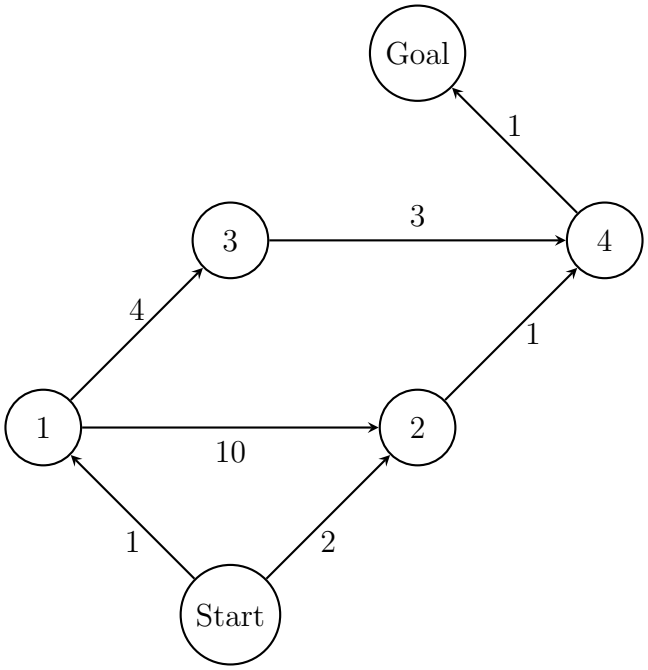

Figure 2: Directed Graph Example

4 Method

4.1 Problem Definition

The problem definition is evaluating whether the Decision Transformer can stitch together training trajectories to output the shortest path from a start node to a goal node in a graph. The shortest path task can be formulated as a Markov Decision Process in which the agent gets a reward of -1 whenever they take an action in a non-goal state and a reward of 0 whenever they take an action in the goal state [2]. The states are the integer labels of the graph nodes (0-indexing) and the actions are the integer labels of the node the agent is going to next. We assume the agent only takes legal actions, meaning there is an edge connecting the current node to the desired next node.

4.2 Data Generation

Three general frameworks were created for dataset generation. The datasets are random walk trajectories from an agent acting randomly in a directed or undirected graph. As described in Section 3.1.1, a trajectory is a sequence of returns-to-go, states, and actions. Each graph and agent's random action are generated randomly from a seed value. We adopt the Decision Transformer's returns-to-go definition of negative path lengths. All graphs were generated and visualized with NetworkX [4].

4.2.1 Framework 1: Replicating the Decision Transformer Graph Setting

Framework 1 recreates the Decision Transformer's graph results. Framework 1's dataset is derived from the original Decision Transformer paper, using a seeded random graph of 20 nodes with a graph density of 0.1 to generate 1000 random walks trajectories of max length 10. 6 different graphs were created from 6 different seeds, resulting in 6 datasets. Random walks were generated by randomly choosing a start node and having the agent traverse to a node by randomly choosing an outward directed edge to take. A trajectory ends when the agent arrives at the goal node which is fixed at node 19, the agent arrives at a node with no outward directed edges, or the trajectory is of length 10. The rewards-to-go are calculated afterwards for each trajectory. During evaluation, the Decision Transformer is asked to generate the shortest path from a random starting node to goal node 19.

4.2.2 Framework 2: Trajectory Segments

Framework 2 begins investigating the Decision Transformer's stitching ability. A directed random graph and an undirected random graph were created using seed 340. Both have 24 nodes with a graph density of 0.09. Two datasets were generated from the two graphs by a random agent traversing the graphs. Each dataset contains 1000 random walks. A max trajectory length of 2 to 5 is randomly chosen for each random walk during generation. Once again, a trajectory ends when the agent arrives at the goal node, the agent arrives at a node with no outward directed edges, or the trajectory is of length 10. The goal node is the node with the most shortest paths of length larger than 5 ending at it. The rewards-to-go are calculated afterwards for each trajectory. During evaluation, the Decision Transformer is asked to generate shortest paths that are greater than length 5 from a random starting node to the fixed goal node. This ensures that the Decision Transformer can never use memory of a shortest path in the training

set as all training trajectories are of length at most 5, forcing it to stitch.

4.2.3 Framework 3: The Bridge Problem

Framework 3 tests the Decision Transformer's ability to stitch and generalize to a novel graph. Two random undirected graphs are created from seed 340. Each graph has 15 nodes and a graph density of 0.2. Each random walk trajectory is contained entirely in one of the random graphs. There is no fixed goal node, so during the trajectory creation, the node a trajectory ends at is the goal node. This is needed for the rewards-to-go calculation. 2000 total trajectories were generated with 1000 trajectories for each graph. During evaluation, a new graph is created by adding an edge between some node in the first graph and some node in the second graph. The Decision Transformer is asked to output the shortest path from a node in the first graph to a node in the other graph while operating in the new graph. This tests the Decision Transformer's ability to stitch by needing to combine together disjoint training trajectories of each of the two graphs to generate the shortest path. This also tests the Decision Transformer's ability to generalize to a slightly novel graph because it has not seen the added edge connecting the original two graphs.

5 Experiments and Evaluations

5.1 Atari Breakout

Atari Breakout results for seed 123 were replicated following instructions from the Decision Transformer's Github. The mean raw score is 65.2 with a standard deviation of 17.4. The original paper reported a mean raw score of 76.9 with a standard deviation of 27.3 across 3 seeds.

5.2 Framework 1 Results

The Decision Transformer has a context length of 10 and is trained for 5 epochs with a batch size of 128. It was evaluated by being prompted to output the shortest path from every node in the graph (except the goal node) to the fixed goal node. Below are the results for seed 324 and 316. The generated graph from the seeds are shown. The histogram shows the proportion of paths to the goal state that had a certain length out of all possible shortest paths to the fixed goal state. The

rest of the graphs are found in the Appendix. The training loss of the Decision Transformer is also provided.

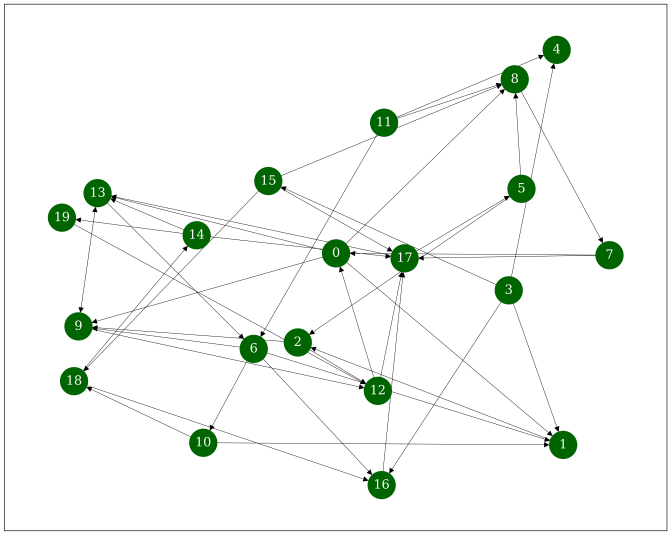

Figure 3: Random Graph from Seed 324. Created with NetworkX [4]

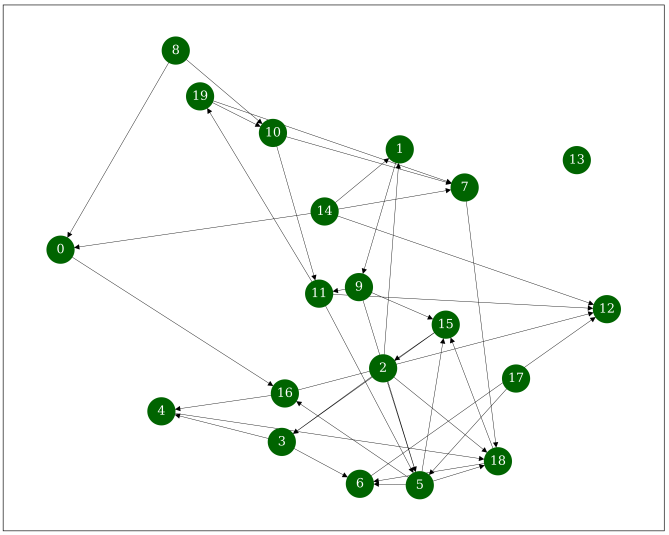

Figure 4: Random Graph from Seed 316. Created with NetworkX [4]

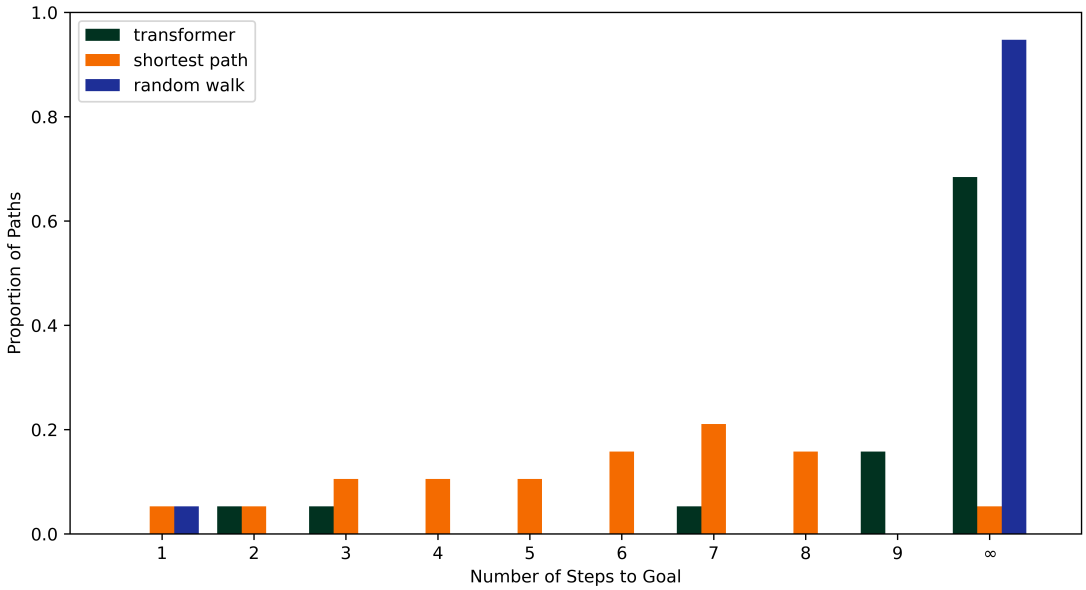

Figure 5: Steps taken to reach the goal node for Seed 324 in framework 1.

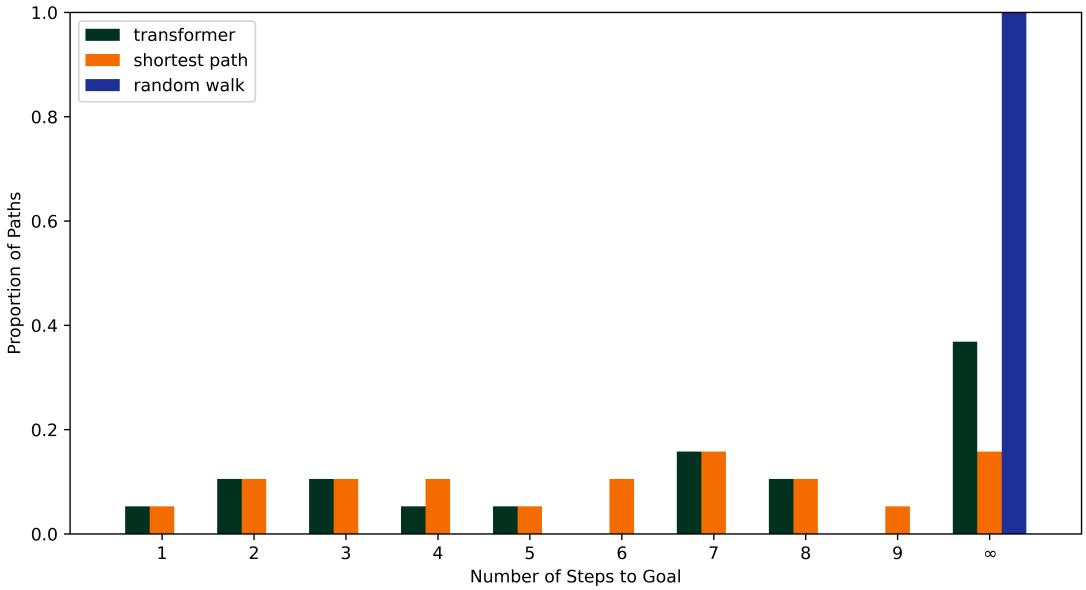

Figure 6: Steps taken to reach the goal node for Seed 316 in framework 1.

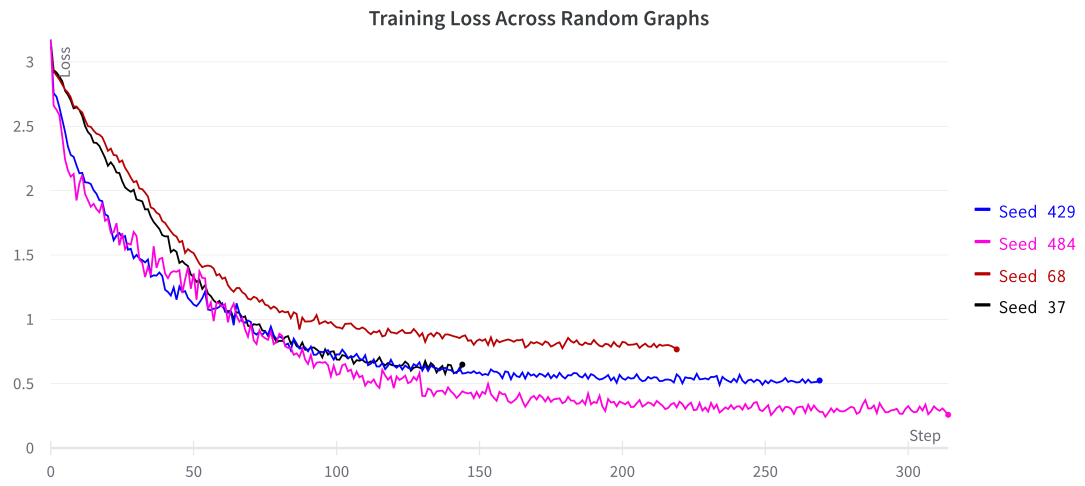

Figure 7: Training Loss for Seeds 429, 484, 68, and 37

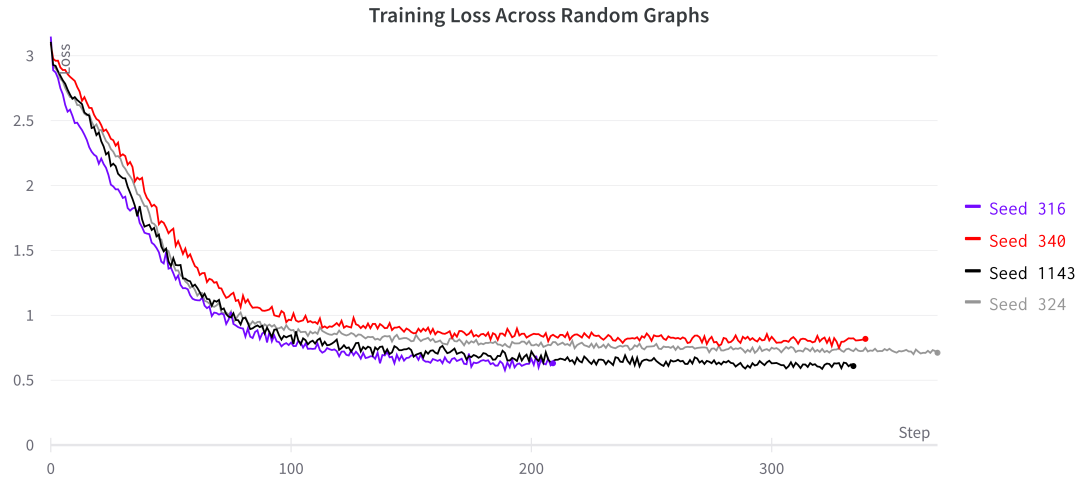

Figure 8: Training Loss for Seeds 316, 340, 1143, and 324

The results highlight that the performance of the Decision Transformer depends on the graph. For example, looking at Figures 6 and 5, both random graphs created from seeds 316 and 324 have long shortest paths. However, the model performs poorly on seed 324 and well on seed 316, highlighting the importance of the underlying graph structure on the Decision Transformer's ability to output the shortest path. On the other hand, the Decision Transformer is almost able to exactly match the shortest path algorithm as seen on seed 429. Overall, the results

are in line with those presented in the Decision Transformer paper, and it shows promising ability to stitch together training trajectories to achieve the goal, while highlighting the influence of the graph structure on the Decision Transformer's performance.

5.3 Framework 2 Results

In Framework 2, the Decision Transformer has a context length of 5 and is trained for 5 epochs with a batch size of 128. Results for the directed graph are shown below.

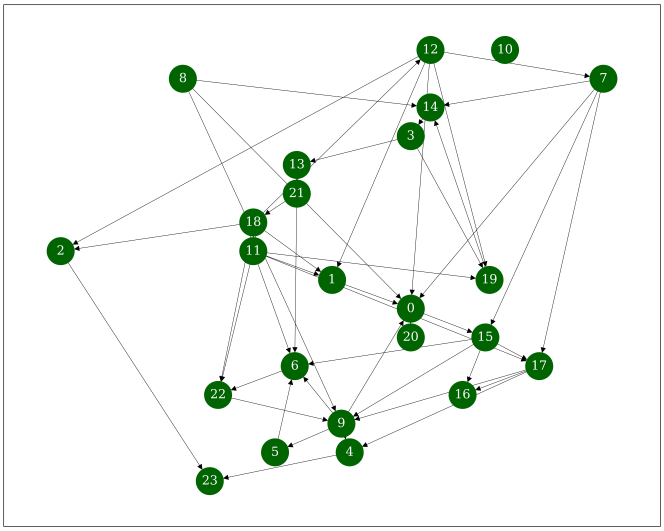

Figure 9: Directed Graph from Seed 340

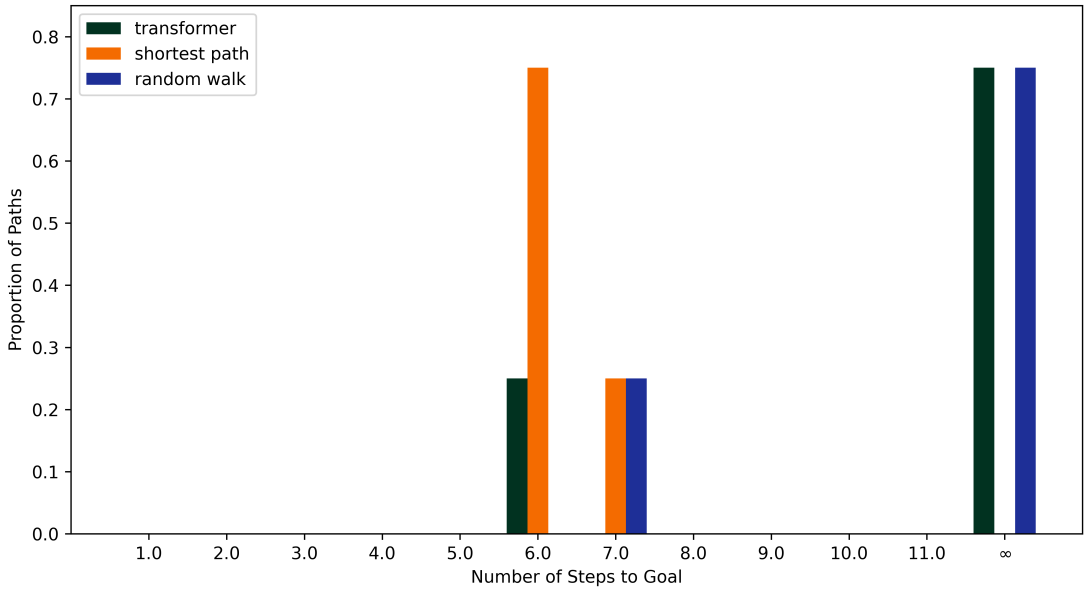

Figure 10: Steps to reach the goal node from 4 starting nodes in the directed graph from Seed 340 using Framework 2.

The Decision Transformer was prompted to generate the shortest path from 4 starting nodes, but the results show that it has a poor ability to stitch. It only correctly output 1 out of 4 shortest paths. To understand these results further, we examine the Decision Transformer's generated paths and the shortest paths:

Generated Paths: (7, 15, 6, 22, 9, 4, 23), (8, 0, 20), (11, 1, 0, 20), (19, 14, 3, 13, 21, 18, 2)

Shortest Paths: (7, 14, 3, 13, 21, 18, 2), (8, 14, 3, 13, 21, 18, 2), (11, 19, 14, 3, 13, 21, 18, 2), (19, 14, 3, 13, 21, 18, 2)

We see that two of the generated trajectories ends with $0 \rightarrow 20$, (8, 0, 20) and (11, 1, 0, 20). This may be caused by training trajectories where a returns-to-go of -4 was associated with the state 0 and the next state was 20. Therefore, when prompted with a returns-to-go of -5 in both the trajectories (because the initial returns-to-go are -6 and -7, respectively), the Decision Transformer remembers associating a returns-to-go of -4 with state 0 and erroneously goes to state 20, thinking it will lead to the shortest path.

I also ran experiments with a context size of 20 and a maximum trajectory limit of 7 instead of 5 to see if that helped the Decision Transformer stitch. Results are similar and are found in the Appendix.

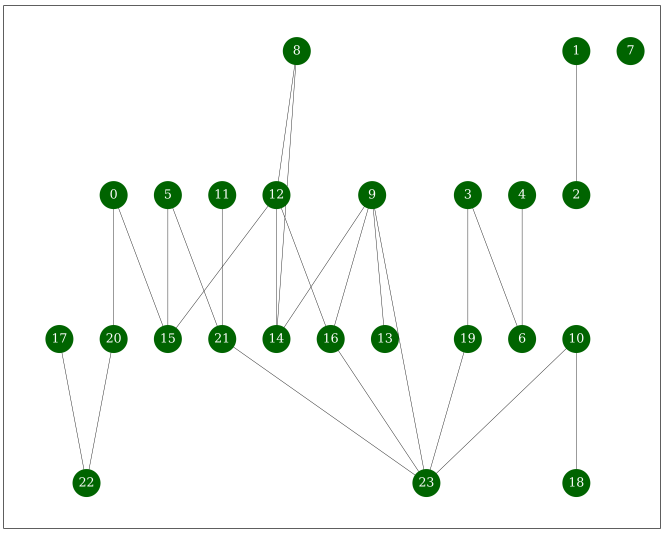

Figure 11: Undirected Graph from Seed 340

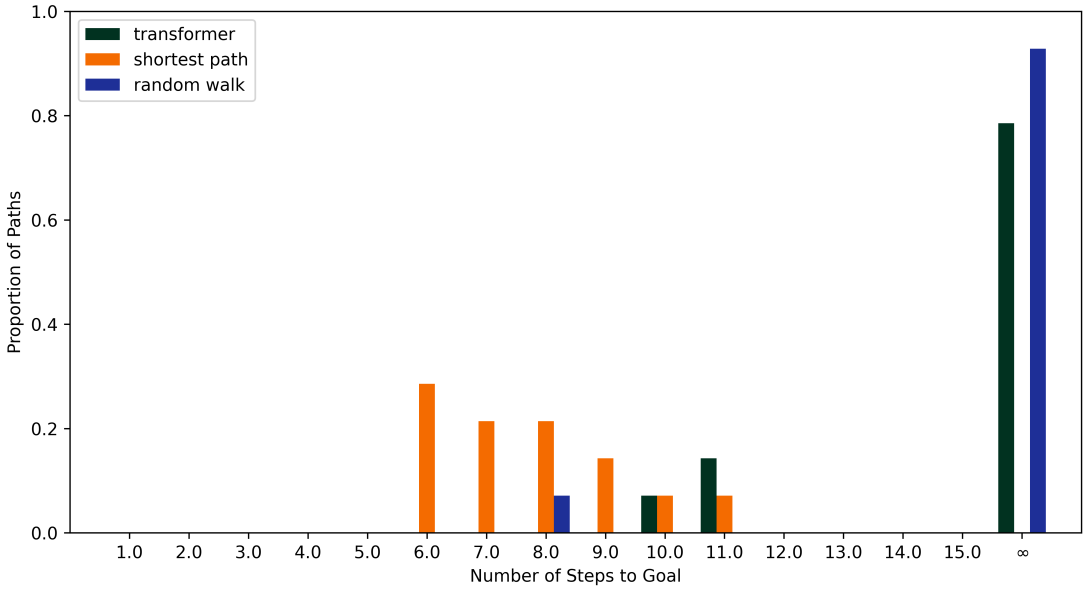

Figure 12: Steps to reach the goal node from 14 starting nodes in the undirected graph from Seed 340 using Framework 2.

From the results of the undirected graph, we see that the Decision Transformer generally fails to stitch. It also generated 0 of the 14 shortest paths. During the evaluation, it generated this path: (4, 6, 4, 6, 4, 6, 3, 6, 3, 6, 3, 6, 3, 19, 3, 19).

This is representative of many of the other generated paths. The common pattern is that the path oscillates between two states. A potential reason for this is that in the undirected setting, the random agent can oscillate between states because the edge is undirected, because there is no sense of direction in the random agent. Looking at Figure 11, at state 4, there is only one action to take and at state 6, there are only two actions to take. Therefore, when the Decision Transformer fails to generalize, it can get stuck oscillating between these two states.

To further investigate this, framework 1 was also used on an undirected graph with 15 nodes and graph density of 0.2. Results are shown in Figure 13. The results were similar to the directed graph setting where the Decision Transformer was able to output the shortest path for 13 out of 14 simulations. It seems that the Decision Transformer fails to stitch as it can output the shortest path when it sees it in the training trajectories, irrespective of whether the graph is directed or undirected.

Meanwhile, there is evidence of the Decision Transformer stitching in a minimal set of generated paths as indicated in Figure 12. Note that the paths of length 10 and 11 are not the shortest paths to the goal node from the starting node in each path as the Decision Transformer generated 0 of the 14 shortest paths. However, the paths of length 10 and 11 indicate that the Decision Transformer was able to reach the goal node but on a longer path, showing it did stitch together training trajectories in a limited number of cases.

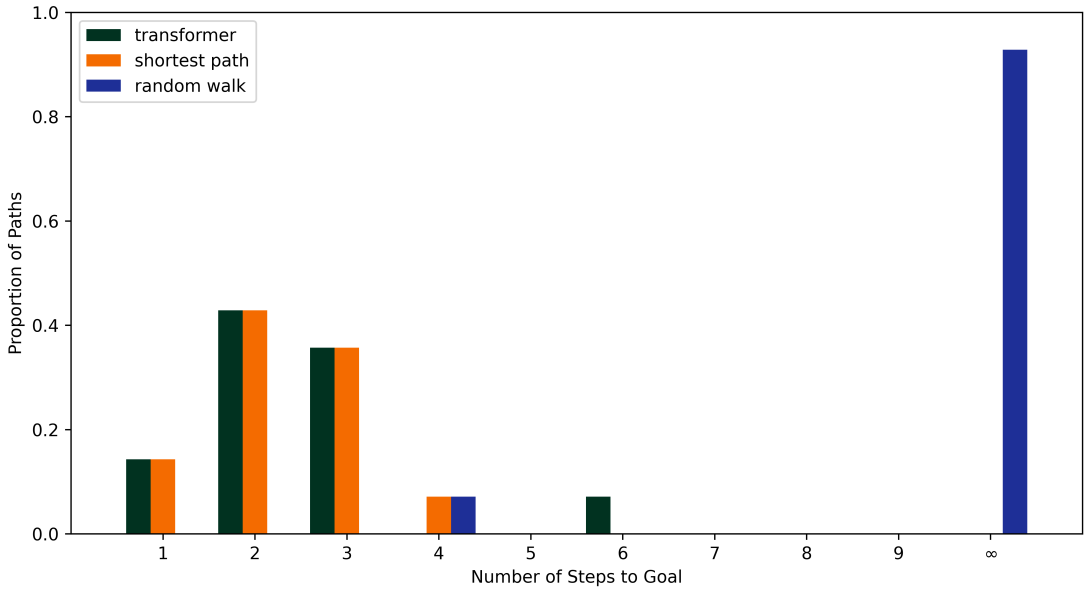

Figure 13: Steps to reach the goal node from 14 starting nodes in the undirected graph from Seed 340 using Framework 1.

5.4 Framework 3 Results

Results from Framework 3 show that the Decision Transformer fails in the proposed setting. The Decision Transformer is unable to generate the shortest path nor stitch a path from the 3 start nodes to the 3 goal nodes. The 3 generated paths were (19, 15, 20, 28, 20, 28, 20, 15, 23, 15, 23), (23, 15, 23, 24, 23, 21, 23, 15, 23, 15), and (28, 19, 28, 20, 16, 24, 23, 24, 23, 15, 23, 15). Looking at them, the Decision Transformer fails to generalize and recognize the added edge because all generated paths lie within one of the original graphs. It does not produce any paths that traverses the added edge and crosses to the other graph. Evaluations where the starting nodes are in the other graph result in the same conclusion. To see if training with more trajectories would help the Decision Tranformer, I also trained with 5000 trajectories. The results are similar and are shown in the Appendix. The visualization of the graphs are also found in the Appendix.

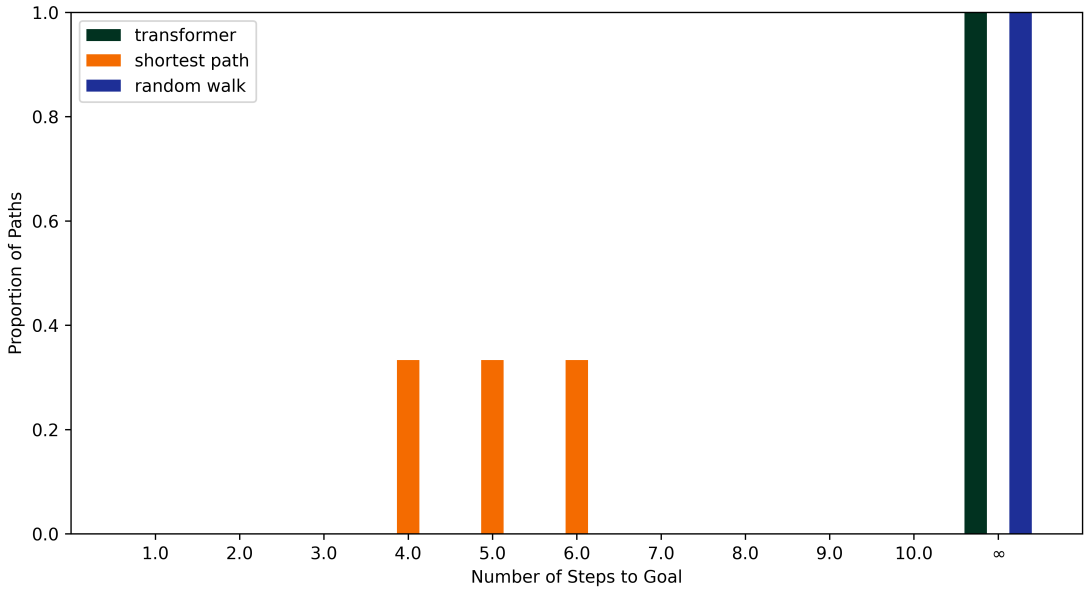

Figure 14: Steps to reach the goal node from 3 starting nodes in the undirected graph from Seed 340 using Framework 3.

6 Conclusion

I investigated whether the Decision Transformer can proficiently stitch together segments of trajectory data obtained from a graph to solve the shortest graph problem. Results show that the Decision Transformer is limited in this capability.

More work should be done on other tasks that require stitching to comprehensively evaluate whether the Decision Transformer can stitch. Future work should also understand the reasons why the Decision Transformer can sometimes stitch together trajectories to reach a goal node, albeit the result is not the shortest path. Furthermore, more work should be done on the bridge setting, specifically seeing if feeding a few trajectories that use the added edge to span the two graphs will result in the Decision Transformer to be able to stitch better. Finally, more controlled experiments in which the graph is not random and the trajectories are ensured to overlap can provide more insights. For example, feeding overlapping trajectories of length 3 like $1 \to 2 \to 3$, $2 \to 3 \to 4$, and $3 \to 4 \to 5$ in a controlled graph to see if the Decision Transformer can stitch in this simpler setting.

Overall, confirming the Decision Transformer's ability to stitch is important as it is relevant in many reinforcement learning settings like transferring learned behaviors

to other tasks. While I examined it in the graph setting, because the underlying architecture is GPT, examining the Decision Transformer's stitching capability will also provide insight into how the transformer architecture can compose things which is relevant in natural language processing. Language is full of composition. I hope this work encourages others to investigate such phenomenon.

7 Code

The code for this paper is available at github.com/iamsamliang/decision-transformer.

8 Acknowledgements

I would like to thank my advisors Shunyu Yao and Karthik Narasimhan for their guidance and support.

9 Honor Code

This project represents my own work, in accordance with the University regulations.
/s/ Sam Liang

References

[1] A. Barreto, S. Hou, D. Borsa, D. Silver, and D. Precup. Fast reinforcement learning with generalized policy updates. *Proceedings of the National Academy of Sciences*, 117(48):30079–30087, 2020. doi: 10.1073/pnas.1907370117. URL https://www.pnas.org/doi/abs/10.1073/pnas.1907370117.

[2] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *CoRR*, abs/2106.01345, 2021. URL https://arxiv.org/abs/2106.01345.

[3] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4RL: datasets for deep data-driven reinforcement learning. *CoRR*, abs/2004.07219, 2020. URL https://arxiv.org/abs/2004.07219.

[4] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using networkx. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[5] M. Janner, Q. Li, and S. Levine. Reinforcement learning as one big sequence modeling problem. *CoRR*, abs/2106.02039, 2021. URL https://arxiv.org/abs/2106.02039.

[6] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *CoRR*, abs/2006.04779, 2020. URL https://arxiv.org/abs/2006.04779.

[7] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020. URL https://arxiv.org/abs/2005.01643.

[8] L. Nie, C. Lin, K. Liao, S. Liu, and Y. Zhao. Unsupervised deep image stitching: Reconstructing stitched features to images. *CoRR*, abs/2106.12859, 2021. URL https://arxiv.org/abs/2106.12859.

[9] M. Pellikka and V. Lahtinen. A robust method for image stitching. *CoRR*, abs/2004.03860, 2020. URL https://arxiv.org/abs/2004.03860.

[10] A. Radford and K. Narasimhan. Improving language understanding by generative pre-training. 2018.

[11] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/ross10a.html.

[12] J. Schmidhuber. Reinforcement learning upside down: Don't predict rewards - just map them to actions. *CoRR*, abs/1912.02875, 2019. URL http://arxiv.org/abs/1912.02875.

[13] A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar, and S. Levine. COG: connecting new skills to past experience with offline reinforcement learning. *CoRR*, abs/2010.14500, 2020. URL https://arxiv.org/abs/2010.14500.

[14] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

10 Appendix

10.1 Framework 1 Results

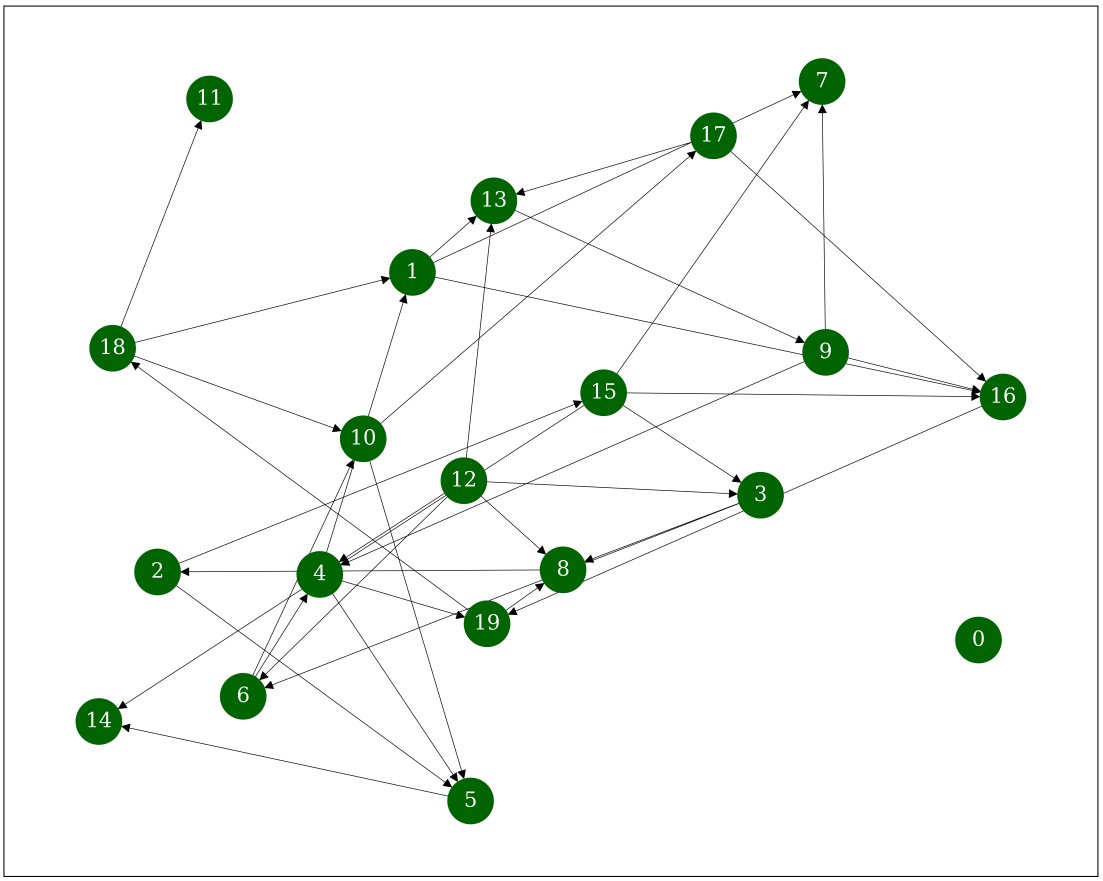

Figure 15: Random Graph from Seed 37. Created with NetworkX [4]

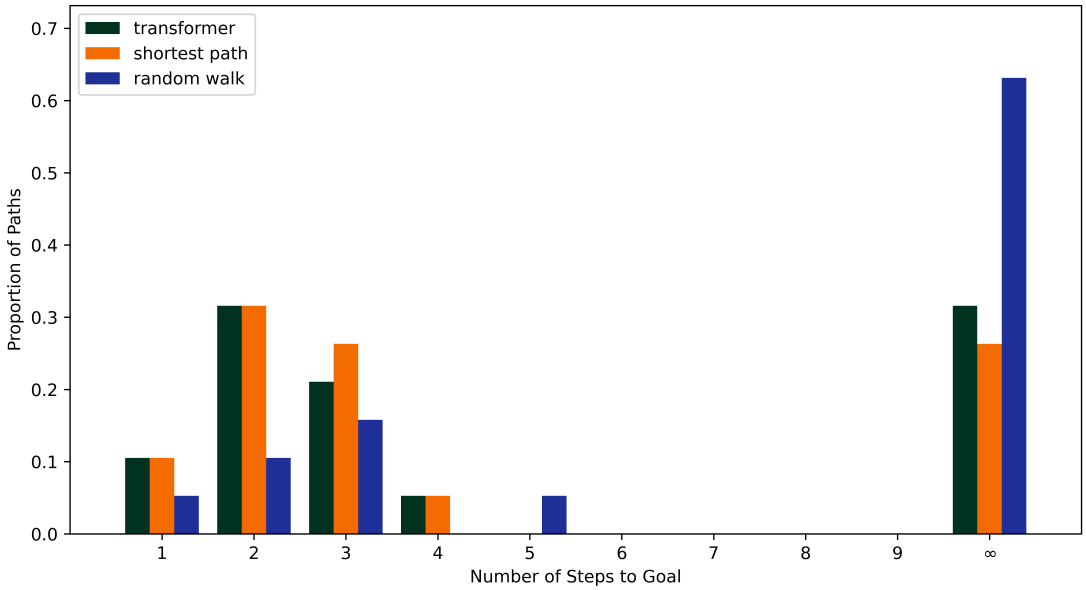

Figure 16: Steps taken to reach the goal node for Seed 37

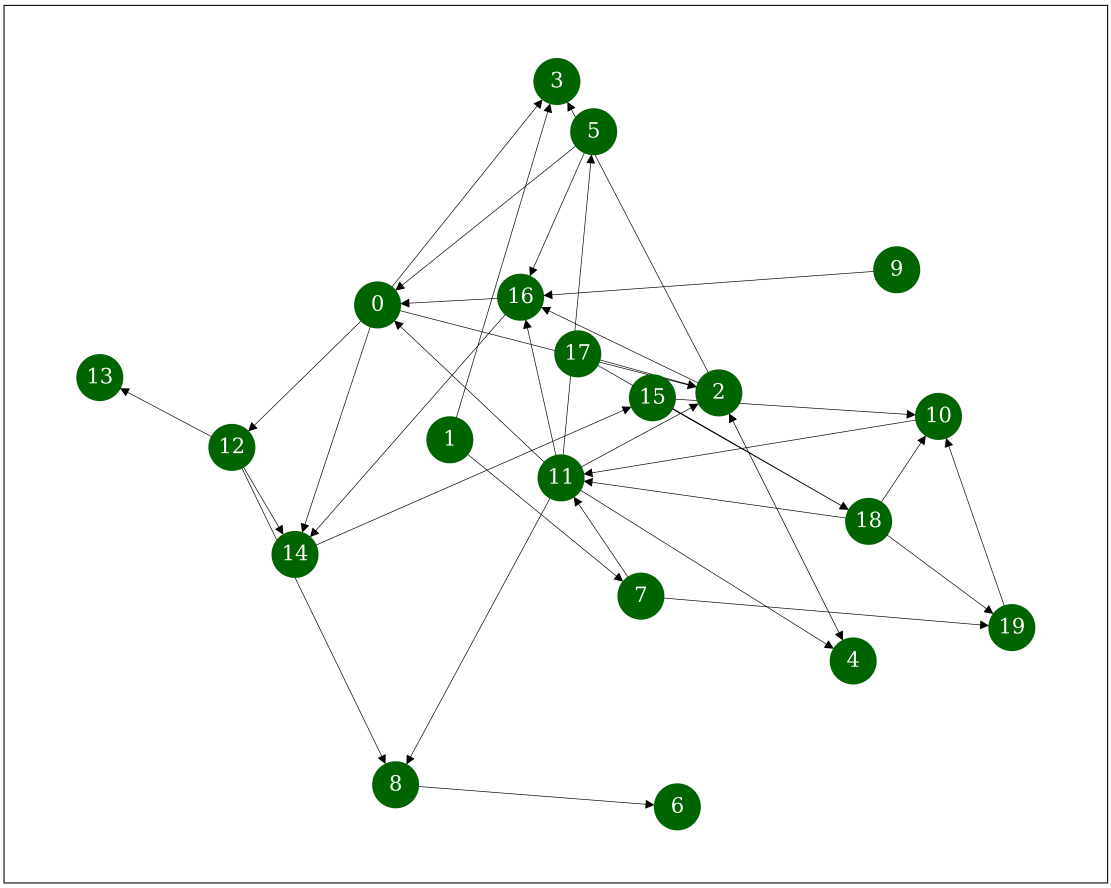

Figure 17: Random Graph from Seed 68. Created with NetworkX [4]

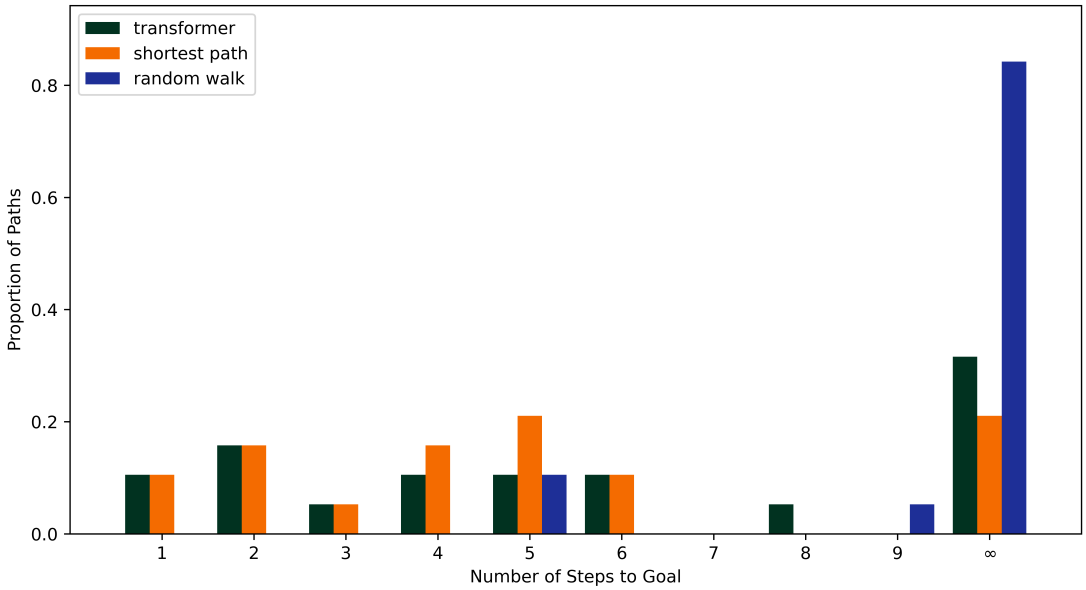

Figure 18: Steps taken to reach the goal node for Seed 68

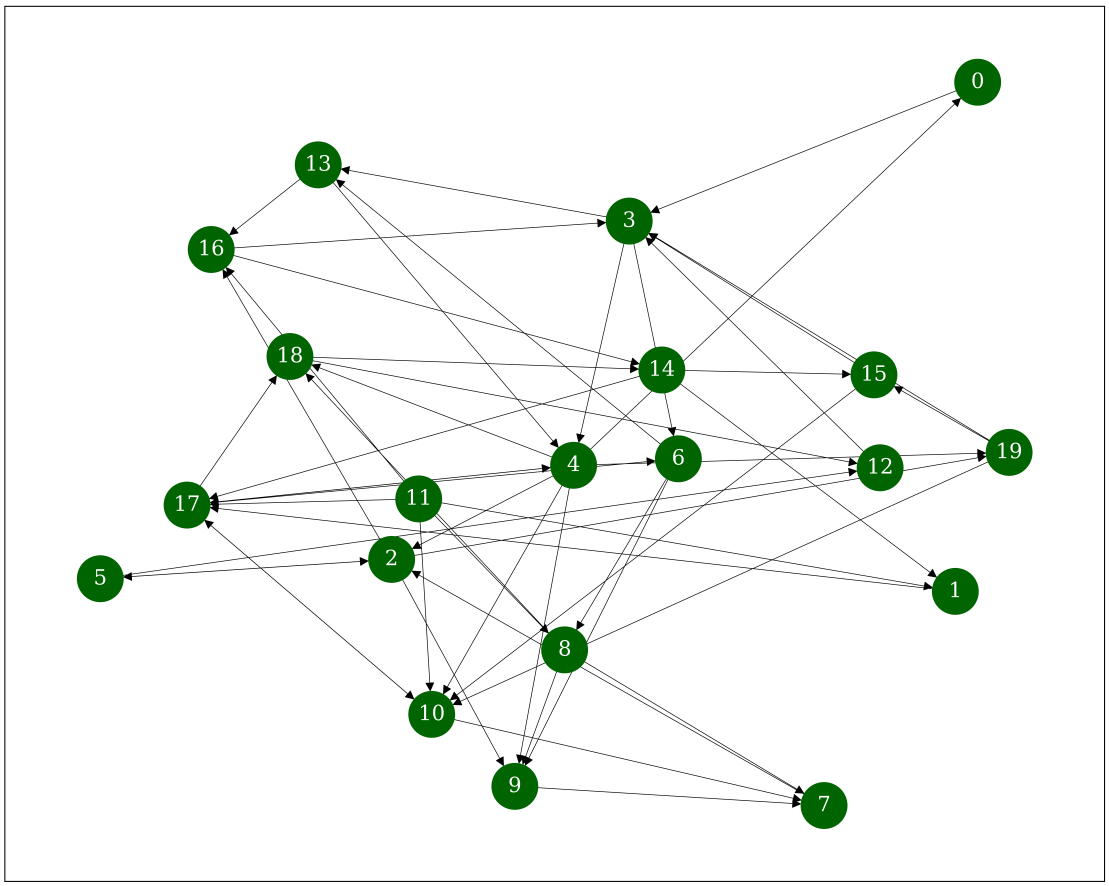

Figure 19: Random Graph from Seed 340. Created with NetworkX [4]

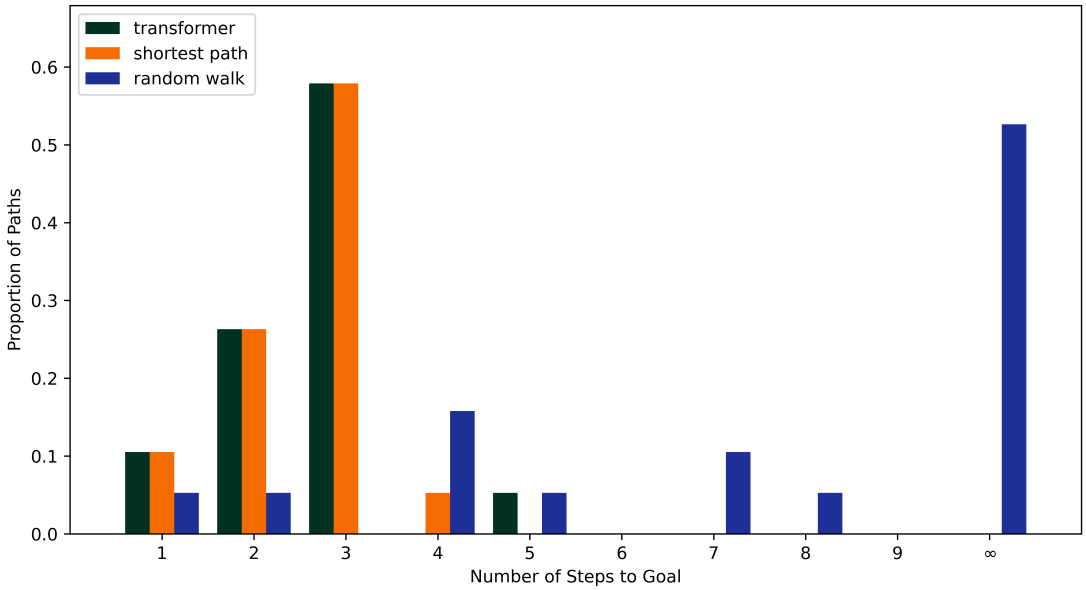

Figure 20: Steps taken to reach the goal node for Seed 340

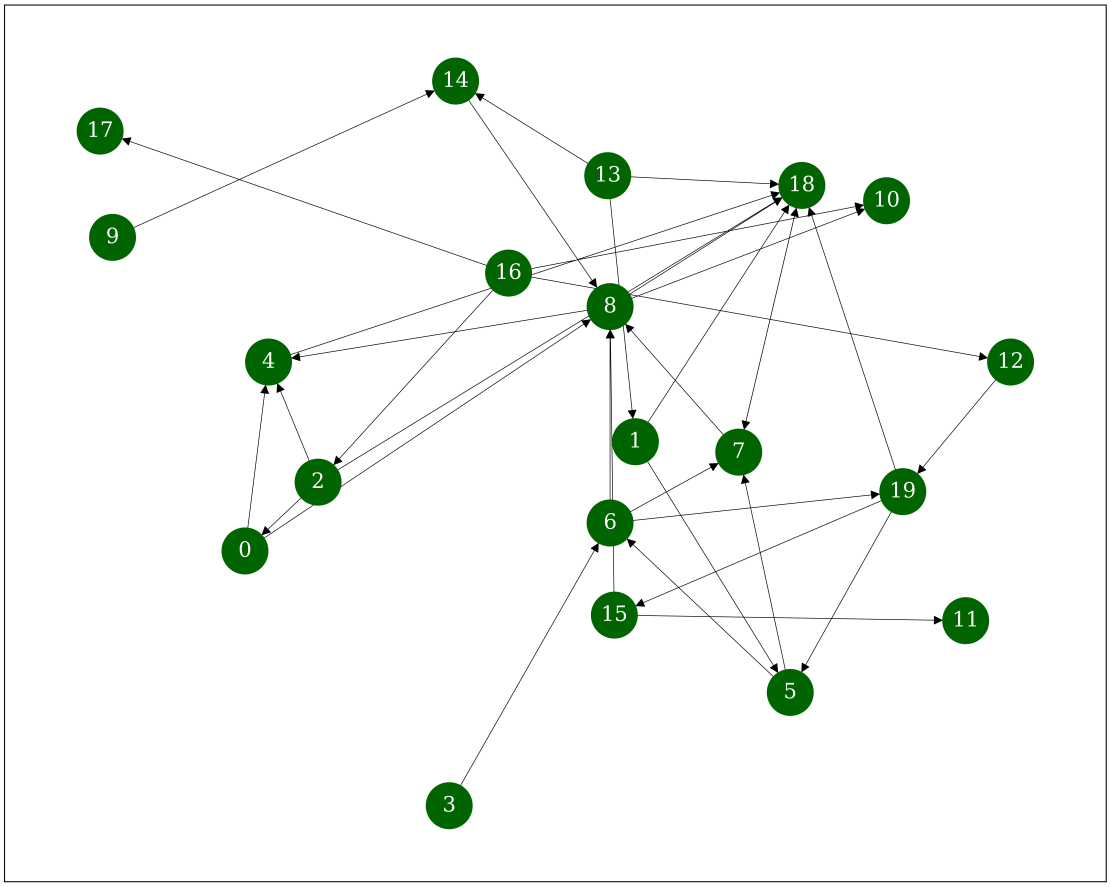

Figure 21: Random Graph from Seed 429. Created with NetworkX [4]

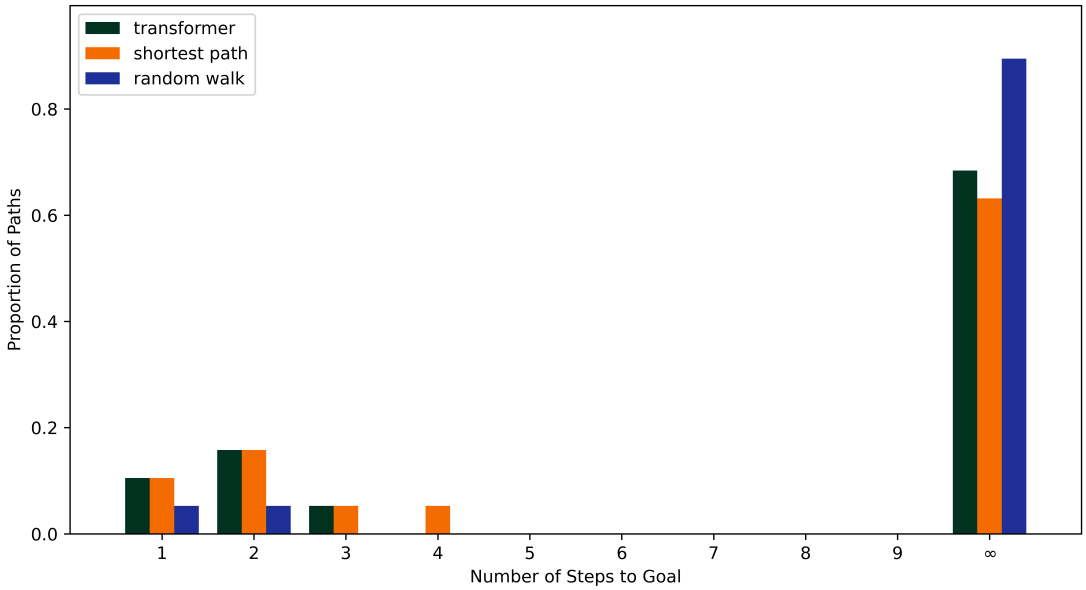

Figure 22: Steps taken to reach the goal node for Seed 429

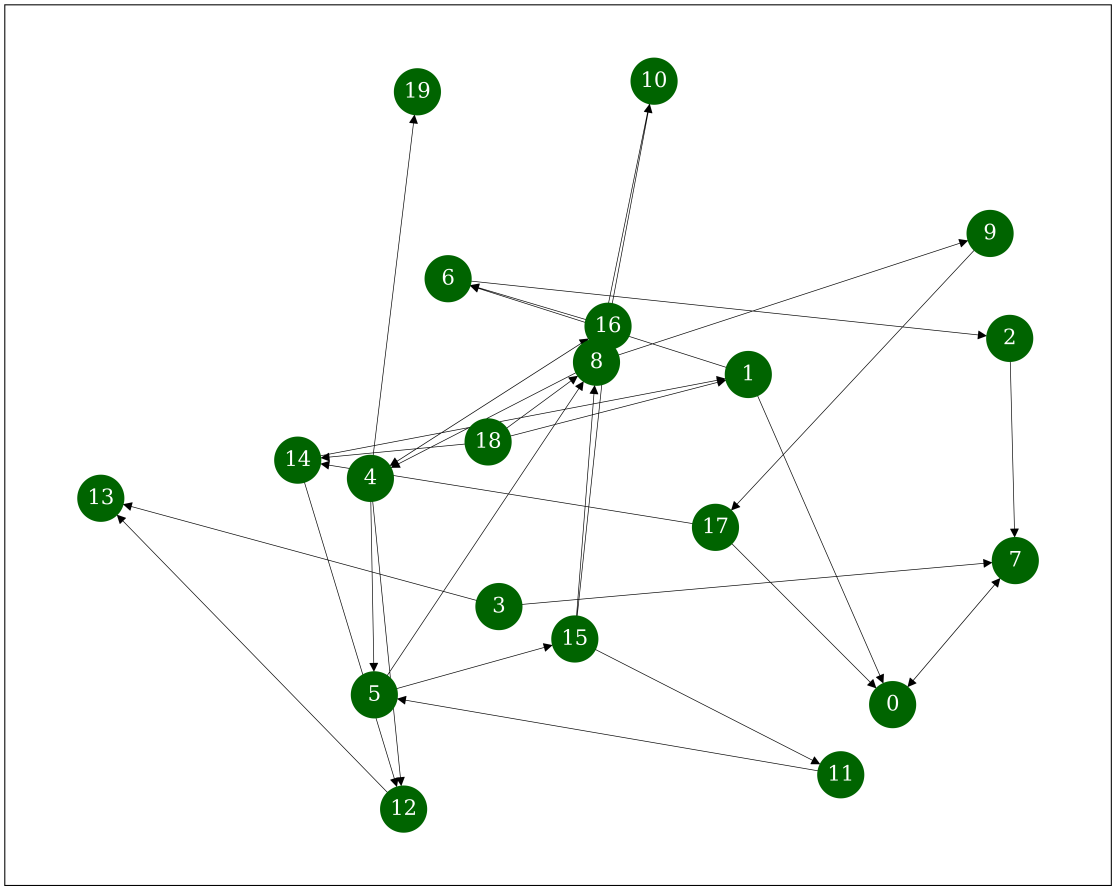

Figure 23: Random Graph from Seed 484. Created with NetworkX [4]

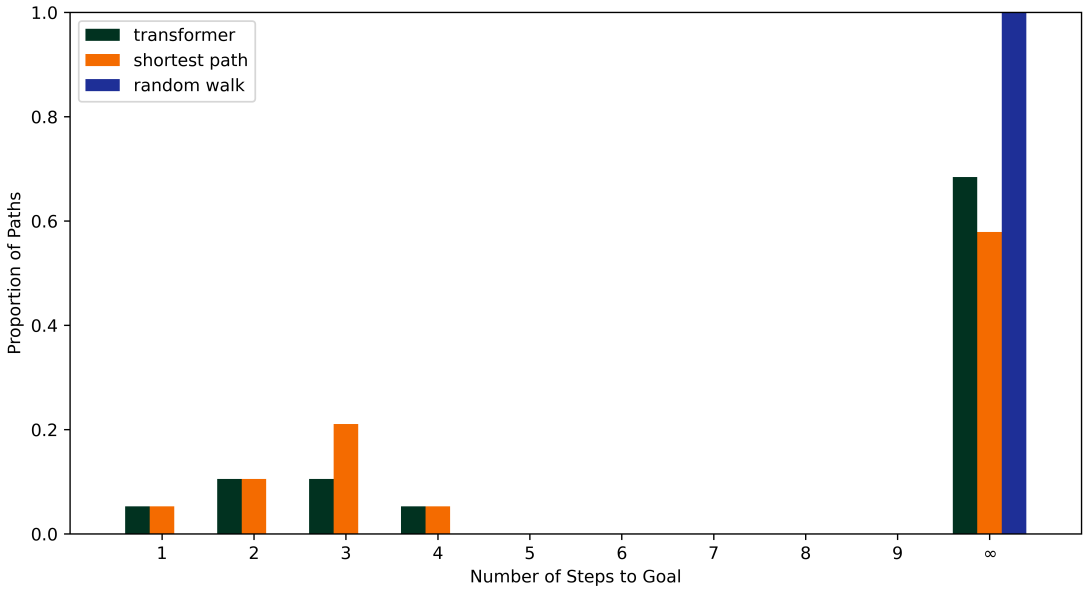

Figure 24: Steps taken to reach the goal node for Seed 484

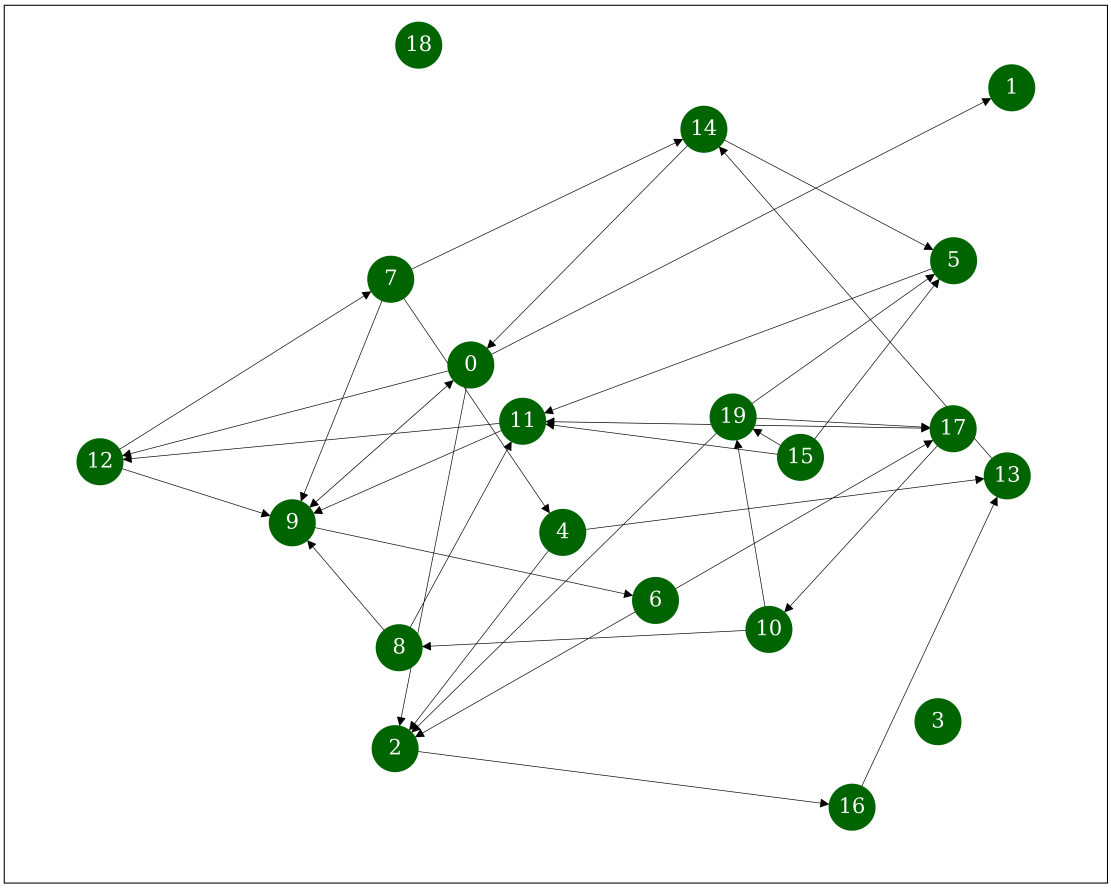

Figure 25: Random Graph from Seed 1143. Created with NetworkX [4]

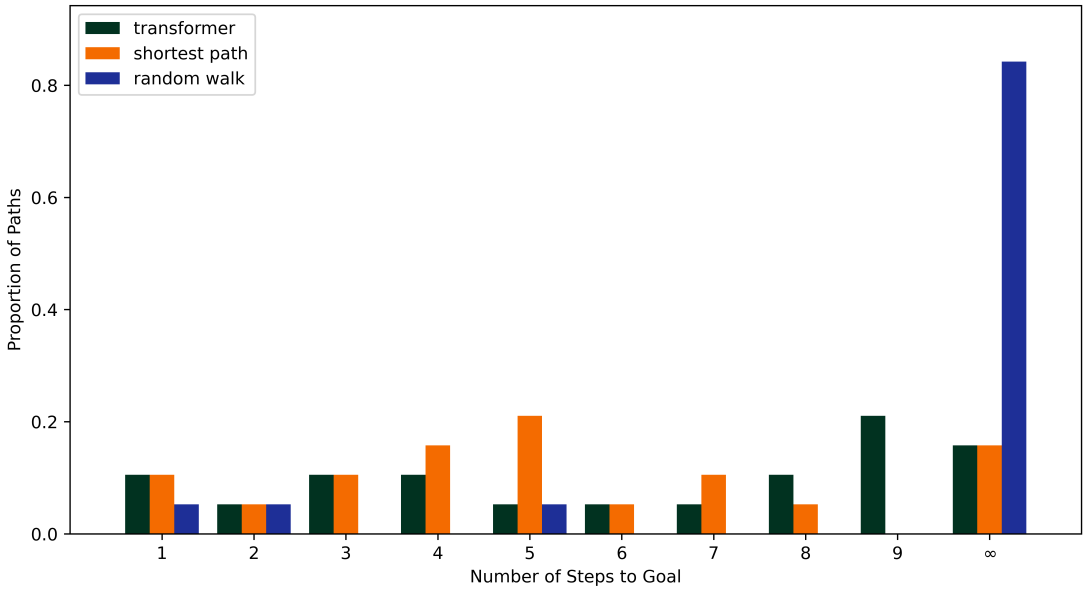

Figure 26: Steps taken to reach the goal node for Seed 1143

10.2 Framework 2 Results

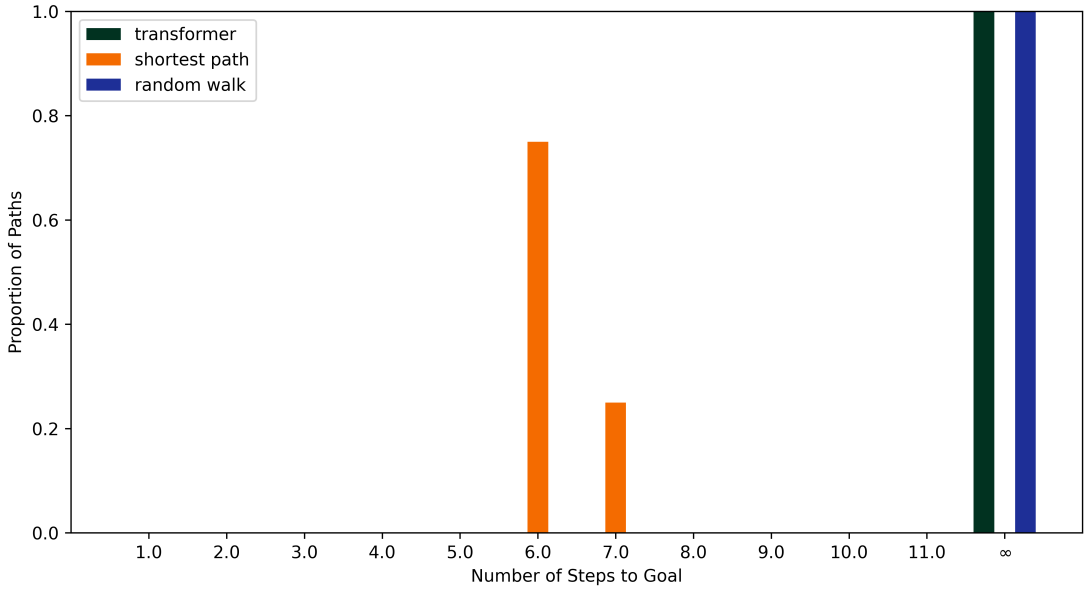

Figure 27: Steps taken to reach the goal node for the directed graph from seed 340 with a context length of 20.

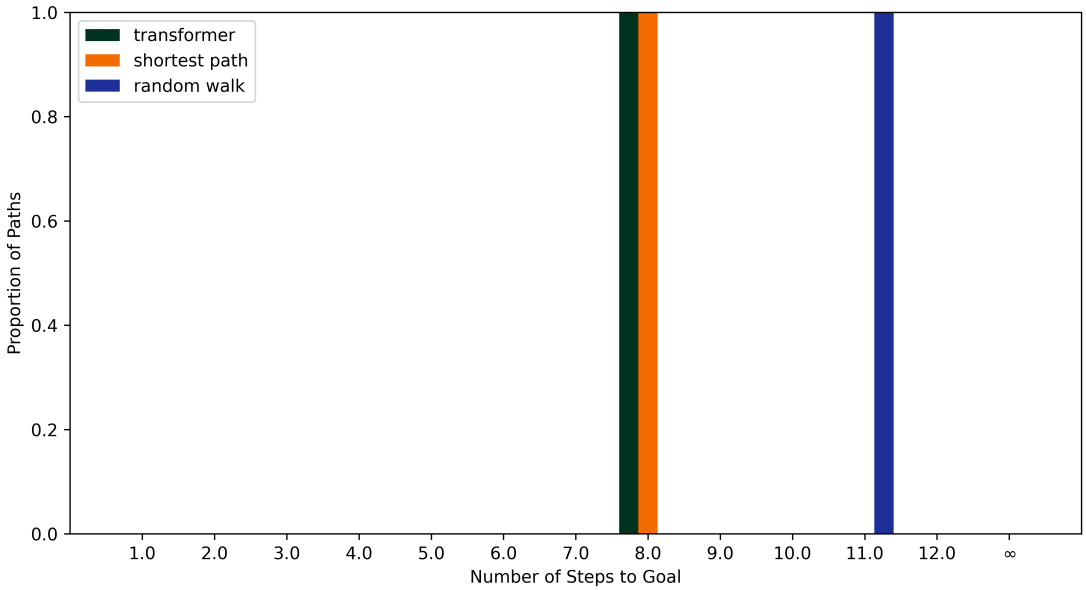

Figure 28: Steps taken to reach the goal node for the directed graph from seed 340 with a max trajectory length of 7. Note there was only 1 such shortest path.

10.3 Framework 3 Results

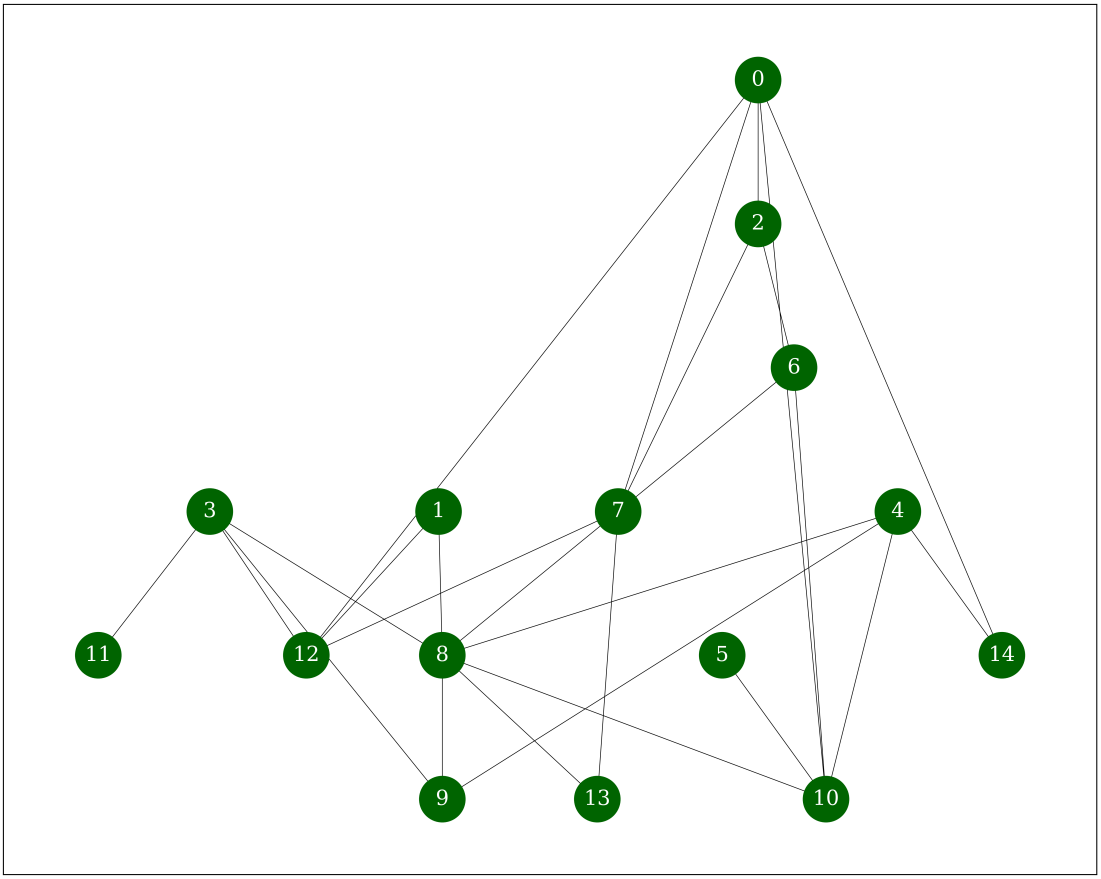

Figure 29: First Random Graph from Seed 340.

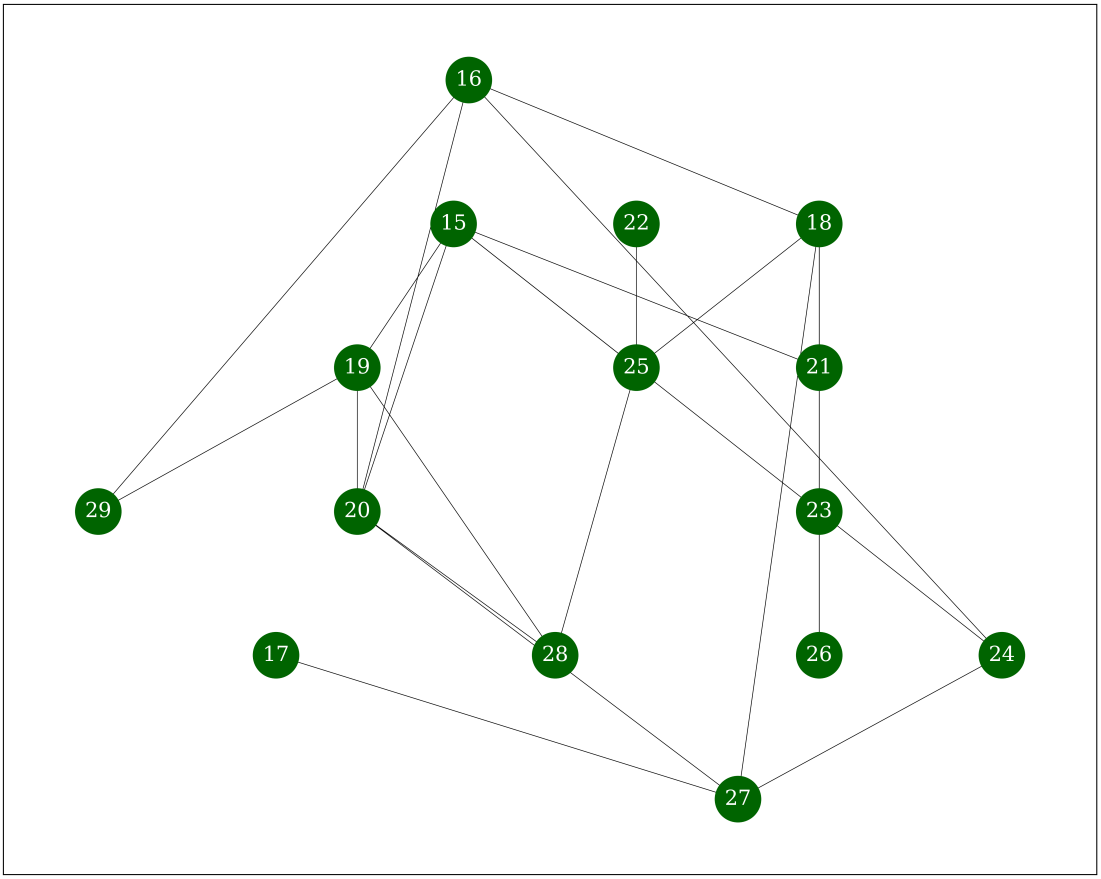

Figure 30: Second Random Graph from Seed 340.

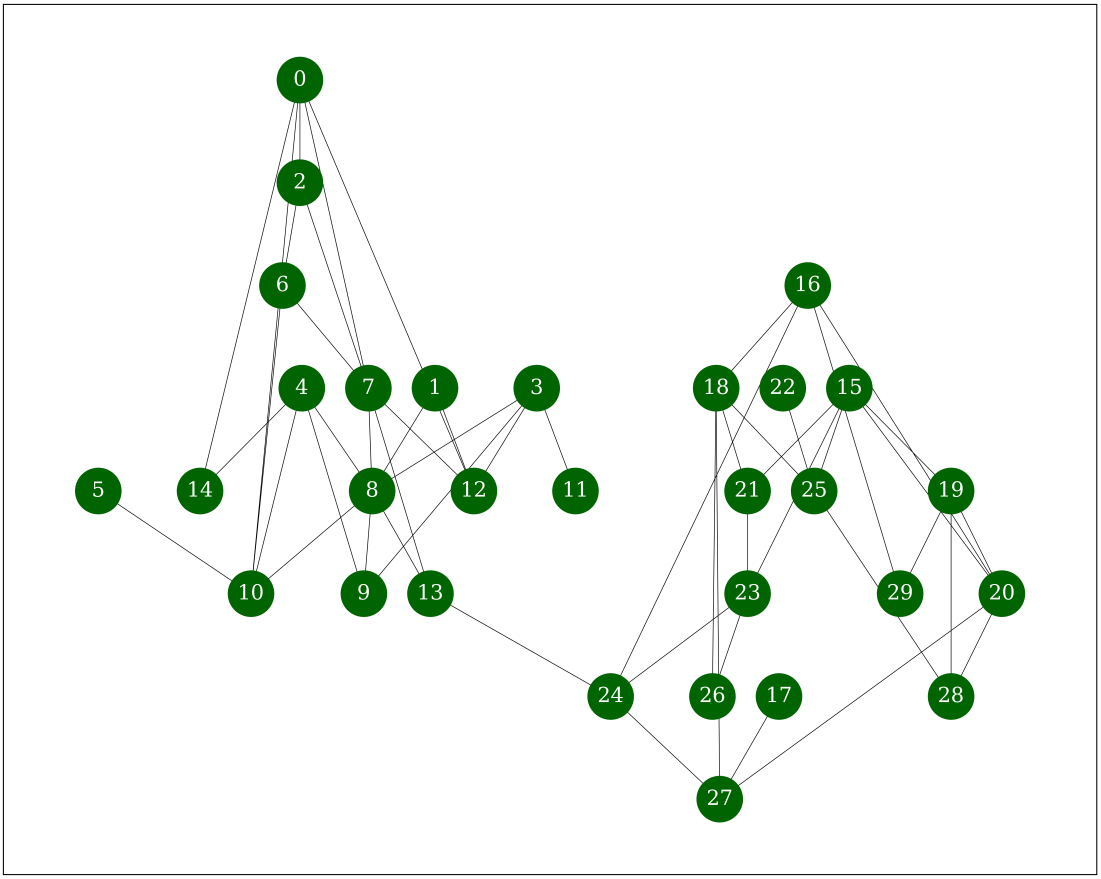

Figure 31: The new graph generated by adding an edge between the first and second random graph.

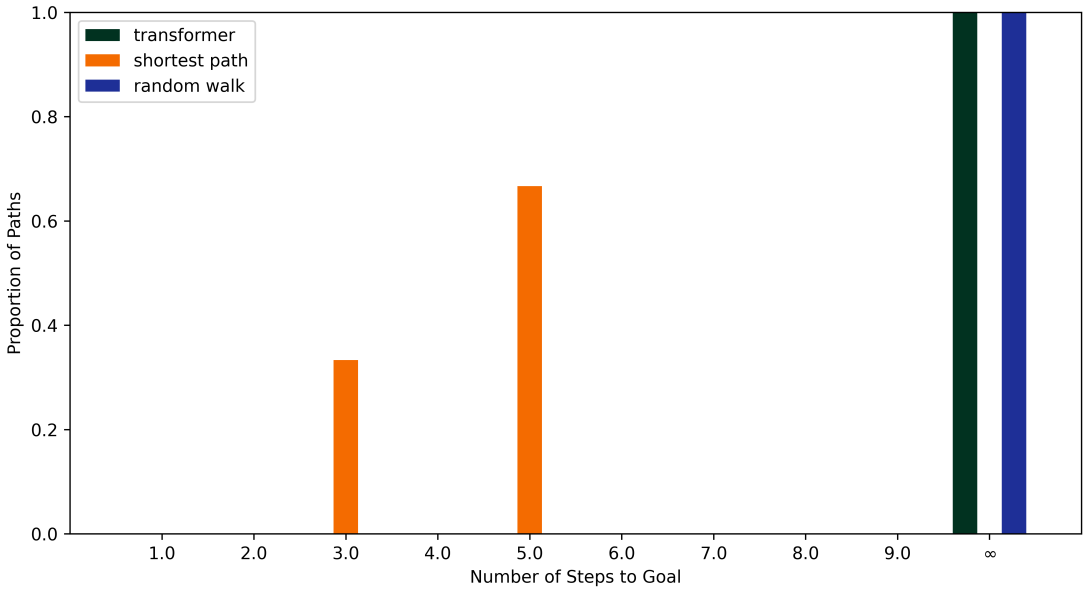

Figure 32: Results when training with 5000 total trajectories at 2500 trajectories per graph.