Reinforcement Learning (RL) Based SFC Request Scheduling in Computer Networks

Eesha Nagireddy

ABSTRACT

This study investigates the use of Reinforcement Learning (RL) to minimize the latency between the source and destination of SFC requests in Neural Networks. This problem gains relevancy owing to Service Function Chaining (SFC) becoming a fundamental concept in modern computer networks to efficiently route and process network traffic through a sequence of specialized network functions. The approach utilizes Deep-Q-Network (DQN) reinforcement learning to determine the shortest path between two nodes using the Greedy-Simulated Annealing (GSA) Dijkstra's Algorithm. The containers within the SFC chain help train the model based on bandwidth restrictions (fiber networks) to optimize the different pathways in terms of action space. Through rigorous evaluation of varying action spaces in models, we assessed the predictive accuracy of each model based on reward-request relationships graphed. A scheduling agent can then manipulate this algorithm to handle maximum scheduling pathways. The findings offer practical implications for reinforcement learning (RL) that can be applied to request scheduling in computer networks to optimize resource allocation, improve the quality of service, and enhance network performance.

INTRODUCTION

Considering the technological boom that has been occurring over the past few decades, day-to-day users are more inclined to yearn for faster network speeds, better resource management, and a seamless integration between the real and virtual world. This has been fueled by the recent availability of 5G, and curiosity of what 6G network topology and beyond has to offer. 5G networks communicate via widespread cellular towers, however their rampant technological delivery has been stunted by a lack of nodes and receiving towers across the globe. This has brought up the question of how to maximize the efficiency these towers have across source-destination SFC chains, a sequence of multiple VNF’s to perform traffic steering chains, in order to remove latency through efficiency. The answer lies within a series of a much larger topology of devices that work to reshape the CPU by accounting for network bandwidth and data harvesting: edge computing, where data travels between nodes within paths. But within this complex topology, how are devices meant to know which server to send a signal to, and vice-versa, to achieve maximal profit? There are many algorithms that may be used to achieve maximal profit, one of which includes Dijkstra’s algorithm. This algorithm was developed by Edsger W. Dijkstra in 1956 and used to find the shortest path through a network topology given a source and destination, however has never been used in combination with Deep-Q-Networking for matrix bandwidth minimization problems [1].

The basis of RL’s application in service based function chains is as follows; the optimization of resource allocation and efficient routing of traffic through short form pathways. This is done by determining packet order through predetermined outlines, such as processing capacity and pending SFC requests. The first step is defining state space, in this context the space would contain information regarding the status of service functions and basis for the current fluctuating

network load. Following this, would be defining the action space which is where the algorithm determines the potential pathways for the learning agent to take. For RL based instructions, these decisions rely on routing decisions made at each interval node, rather than the more common and less efficient Random Walk with Restart (RWR) application. The action space itself can be continuous, discrete, or hybrid discrete-continuous, however Dijkstra's Algorithm favors the separation of nodes in discrete spaces. Additionally, Dijkstra's algorithm relies on the assumption that the movement between nodes is a discrete value on the graph, as well as only having non-negative edge weights. Potential negative weights when factored into the algorithm, would skew the data and favor those nodes over others despite there being no correlation between negative edge weights and shorter pathways.

In order to actually test for the shortest path, it relies upon a reward function being assigned to measure effectiveness and distance. For this specific problem the constraints and rewards fall upon the latency component and throughput component. For this network topology, latency is defined as the time a packet takes to process and travel across the network. Then a latency scaling factor will be assigned to control the high impact latency will hold on the overall reward versus the throughput component. Following iterative refinement, these processes apply the Dijkstra's algorithm as a RL based model versus policy based.

RESULTS

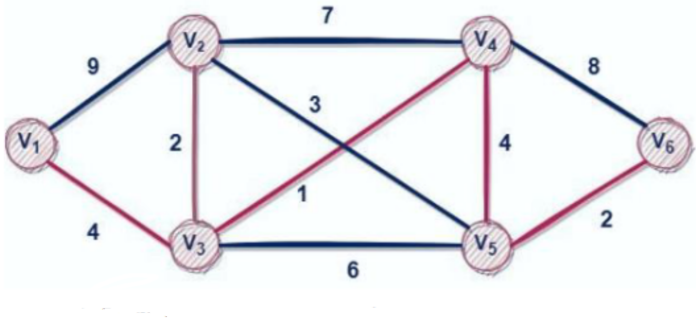

Figure 1: University of Wisconsin Example of Network Topology

The main goal of Dijkstra's algorithm is to achieve the best path from a source to the destination with minimal cost [1]. In this case, cost refers to the distance between various nodes. In Figure 1, V1 is the source and V6 is the destination. To determine the shortest path, it is helpful to create a table to keep track of the distances. This topological mpa is a simplified version of VFN multi-domain SFC orchestration diagrams, illustrating the VFN lifestyle as it iterates through a status monitoring node. Similarly, table initialization gives Dijkstra's method the context and starting point it needs to carry out iterations successfully. It guarantees that the method can determine the shortest path connecting the source node to every other node in the network through distance. By default, setting the distances to infinity will allow the values to be updated through each iteration. It would also be helpful to create a list of the visited and unvisited nodes. By setting all nodes as unvisited initially, the algorithm ensures that it considers all nodes for potential exploration. Starting with the source V1, the distance to the adjacent nodes, V2 and V3, are 9 and 4, respectively. We then update the table to include these values, and mark V1 as visited to ensure it does not get counted as an adjacent neighbor node again. Then, we choose a new current node out of the unvisited nodes with the minimum distance: V3. Again, we calculate the distance to all the unvisited neighbor nodes and update the shortest distance if it's shorter than the old distance. The distance to V2 is 6, V4 is 1, and V5 is 6. We also add V3 to the visited nodes list.

This process repeats until termination, of which is when there are no unvisited nodes left or when there is no node with a tentative distance less than infinity. This means that all nodes reachable from the source node have been visited. The completed table should look similar to the tabulation below. The last step to finding the shortest path is to retrace the path starting from the destination to the corresponding previous node. This program would run until completion, and would be rewarded when a choice leads to a shorter pathway. It is able to do this because Dijkstra's algorithm is greedy, meaning an agent will travel to the nearest vertex. Essentially making the shortest and most optimal transversal steps in order to translate to the shortest possible global output. Thus, the shortest path is V1-V2-V4-V5-V6 with a total cost of 11, but not just for individual decisions across different nodes but the optimal path for the network's topology (going from source to destination).

Node	Shortest Distance	Previous Node
V1	0	
V2	9	V1
V3	4	V1
V4	∞	
V5	∞	
V6	∞	

Node	Shortest Distance	Previous Node
V1	0	
V2	6	V3
V3	4	V1
V4	1	V3
V5	6	V3
V6	∞	

Node	Shortest Distance	Previous Node
V1	0	
V2	3	V5
V3	4	V1
V4	1	V3
V5	4	V4
V6	2	V5

DISCUSSION

Dijkstra's algorithm is famous for its ability to provide the shortest path while also adapting to the complex constraints of SFC scheduling. This algorithm is a Greedy-Simulated Annealing(GSA) algorithm that was initially chosen because of its two step approach, allowing for higher contrast on the basis of their heuristic framework [2]. However, successful implementation requires consideration of factors such as edge weights, dynamic network conditions, and real-time adaptability. Its comprehensive approach and meticulous journey provide a powerful mechanism

for optimizing SFC scheduling in complex network topologies, ensuring efficient service delivery and resource utilization. Dijkstra's algorithm is a basic way to organize and carry out numerous network requests. This algorithm involves comparing the cost required to connect to adjacent nodes and thus resulting in the shortest path that is the most cost efficient. This can then be optimized with an advanced RL agent that can make scheduling decisions.

Finding the shortest path from a specific source to a specific destination is an example of just one request a potential user may have. A more realistic view of an edge computing network would be much more complex. Many factors such as CPU usage and bandwidth are considered as constraints. Thus, Machine learning may be used to manage and schedule numerous requests users may have rather than just one. Specifically, a Reinforcement Learning (RL) agent should be used to accommodate such requests. The RL framework displayed here can suitably work with a given number of CPU cores, bandwidth, and compute the shortest path using Dijkstra's algorithm. Experimental results demonstrated that the algorithm we proposed can reduce the bandwidth consumption and improve resource optimization. In future studies, owing to the encroachment of new 5G, 6G, and intel processors; RL based machine learning is expected to be deployed in SFC request scheduling networks.

METHODOLOGY

We will now analyze the current implementations of RL algorithms on schedule based pathways, specifically the various methods of approach. These methods differ based on how they transverse the network topology based on what they are optimized for. Then these methods will be compared to the short form pathfinding found in Dijikstra's algorithm.

CURRENT SCHEDULE BASED PATHWAYS

Shortest Remaining Time First (SRTF) algorithms are most commonly used for SFC based Scheduling requests, however there are many downsides through implementation. SRTF algorithms are the preemptive form of SFJ algorithms. The nodes and pathway for the packets are determined by the agent's evaluation of the burst time. Burst time is also known as the execution time, which is the amount of time it takes the CPU to process an input. However when compared to Dijkstra's algorithm, process starvation occurred sooner in the SRTF algorithm [2]. Essentially the SRTF algorithm would prioritize short form pathways over and long term topological decisions. Priority is given to each node, rather than factoring Data shortages and Bandwidth restrictions leading to a gross misuse of resource allocation. Despite its benefits, this would be the incorrect method for scheduling requests because the reward metrics could not utilize scalability with flow rates[3].

Multi-Objective Shortest Path Algorithms are a similarly quick short path algorithm, however for this specific problem type the advantages of said algorithm hold little significance. Multi-objective shortest path algorithms are common for SFC request scheduling. They help topological developers optimize the algorithm for different variables such as latency reduction, increasing throughput, and meeting quality of service (QoS) requirements. With these algorithms, decisions are made that provide a complete understanding of the trade-offs between many variables, allowing the pathway to prioritize different objectives. However, they have

some disadvantages because multi-objective algorithms are often more complex than single-objective algorithms, they have higher thresholds in order to actually maintain the software [4]. Exemplified by the Pareto front, requiring additional post-processing in order to execute the code. Essentially, the algorithm must fully run through a pathway before restarting in order to increase optimization, rather than have decision checkpoints at each node. Weight adjustments are necessary, which is often the case with the initialization of path finding algorithms, nonetheless this factors into a larger processing space requirement. Despite these challenges, multi-objective shortest path algorithms provide a versatile and adaptive solution to complex planning problems, making them a useful tool.

DIJKSTRA'S ALGORITHM

Dijkstra's algorithm, a fundamental tool in network pathfinding, provides a systematic approach to finding the shortest path in a weighted graph. In the context of Service Function Functions (SFC) it begins with graphing the topology, with network nodes and edges (what is constrained by bandwidth and latency issues). Beginning by simulating the environment using Java. Adopting figure 1 as a small scale network instance for physical networks of SFC deployment. In this paper the assumed model contains 6 nodes, so I chose a machine with an i7 CPU and accorded RAM. During initialization, the algorithm sets non-negative conditions, specifying the distance between the source node and itself to be 0, while all other distances are initially set to infinity. The visited nodes record starts with an empty list. At the heart of the algorithm, an iterative process, goes as follows: at each iteration, an unvisited node with the smallest expected distance (lowest cost) becomes the current node.. If the neighboring node’s temporary distance is less than its recorded, the table is updated and the path is altered.

The algorithm continues until all nodes have been visited or the destination node (last node of the SFC query) is visited. Once finished, path rebuilding follows to determine the shortest path from source to destination. During this process, constraint checking ensures that the criteria of the SFC request, such as the sequence of service functions and any quality of service (QoS) requirements, are met.

ACKNOWLEDGMENTS

I would like to thank Congzhou Li, my mentor during the summer research project. I could not have done this without him, along with the rest of the research team.

REFERENCES

[1]D. Rachmawati, L. Gustin. (2020). Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem. *Journal of Physics: Conference Series,* **1566**, 26-27. doi: https://doi.org/10.1088/1742-6596/1566/1/012061

[2]Y. Wu, J. Zhou. (2021). Dynamic Service Function Chaining Orchestration in a Multi-Domain: A Heuristic Approach Based on SRv6. *National Library of Medicine, 21* (**19**), 26-27. doi: https://doi.org/10.3390/s21196563

[3]T. O. Omotehinwa. (2022). Examining the developments in scheduling algorithms research: A bibliometric approach. *Heliyon, 5* (**8**), 9510. doi: https://doi.org/10.1016/j.heliyon.2022.e09510

[4]S. Zheng, C. Zheng and W. Li(2022). Research on Multi-objective Shortest Path Based on Genetic Algorithm. I*nternational Conference on Computer Science and Blockchain (CCSB)*, **2** , 127-130. doi: https://doi.org/10.1109/CCSB58128.2022.00030