PES
UNIVERSITY

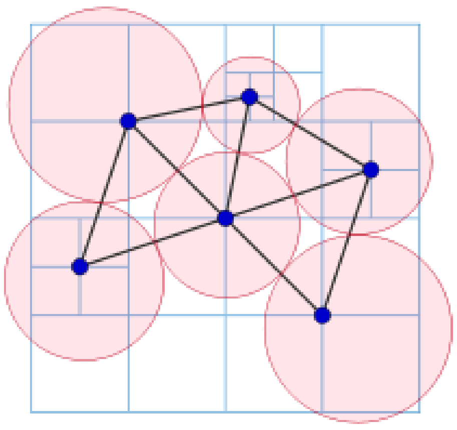
ADVANCED ALGORITHMS PROJECT

<u>**NAME**</u>**: SHUBHA V HEGDE**

<u>**SRN**</u>**: PES1UG20CS419**

<u>**SECTION**</u>**: G**

Optimized Shortest Path-Finding Graph algorithm using Graph Neural Networks



Problem Statement:

Given a question about the shortest path between two points A and B (example city Bengaluru and Delhi) and a graph (as a set of nodes and edges), an algorithm developed to return an answer after learning the function.

Objective:

In depth analysis of graph algorithm to find shortest path between points using an algorithm similar to Dijkstra's,Bellman-Ford, Floyd's, and how Graph Network with attention read and write can perform shortest path calculations with minimal training and 100% accuracy.

Expected Results:

Given the question "How many stations are between city Bengaluru and Pune ", the correct answer should be "6".

We also look at works that use Graph Neural networks trained with Graph-Question-Answer tuples. Each tuple contains a unique randomly generated graph, an English language question and the expected answer.

CLASSICAL ALGORITHMS FOR SHORTEST PATH FINDING PROBLEM - A*, Prim’s, Djikstra’s, Floyd.

→ Graphs can be utilized for various purposes, having applications in various scopes.

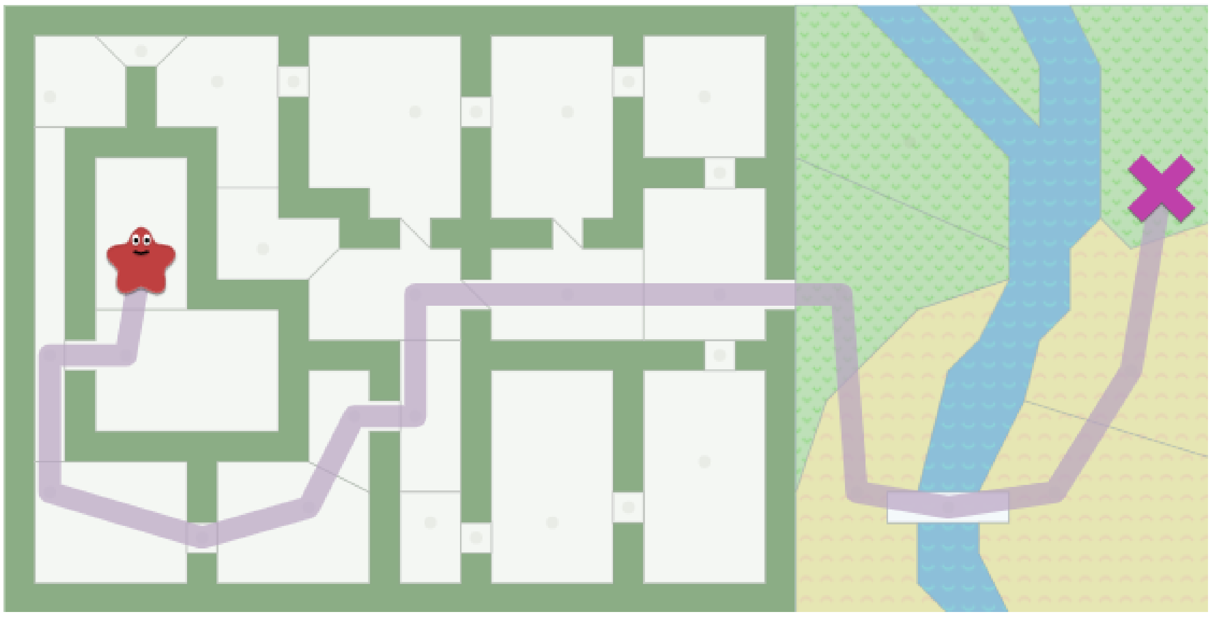
Real-life Problems can be modelled using Graphs, with nodes containing information about the type of data, and edges containing information about relative weights or the relation between the different data points. One such example can be the real-life problem of Fluid Particle Simulation, where multiple graphs are generated – nodes containing information about the type of particle (Eg: Water), it’s velocity, and edges containing information about relative distance between the different particles and displacement from the container boundaries – to learn the dynamic information between the particles in order to simulate the dynamic fluid motion of the particles using Graph Neural Networks.

The most used application of Graphs is the Shortest-Path finding problem. In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

Common Applications of Shortest-Path Problems in Real Life

- Transporation problems (with solutions like Google Maps, Waze, and countless others) are a prime example of real-world applications for shortest path problems.

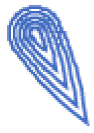
- Motion Planning - Flip book animation is like a shortest path problem. At each page of the flip book, you're using the path of the limbs to anticipate the next frame.
- Communication Networks - Fastest way to send an email, best place to provide a CDN (Content-Delivery Network) of your images and JavaScript.



Algorithms which can be used for solving this problem are ***A* algorithm, BFS, Djikstra's Algorithm, and Floyd-Warshall Algorithm.***

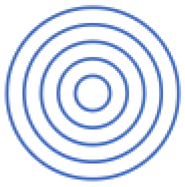
A* is a modification of Dijkstra's Algorithm that is optimized for a single destination. Dijkstra's Algorithm can find paths to all locations; A* finds paths to one location, or the closest of several locations. It prioritizes paths that seem to be leading closer to a goal.

Complexity: **O(bd)**



Breadth First Search explores equally in all directions. This is an incredibly useful algorithm, not only for regular path finding, but also for procedural map generation, flow field pathfinding, distance maps, and other types of map analysis.

Complexity: **O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used**



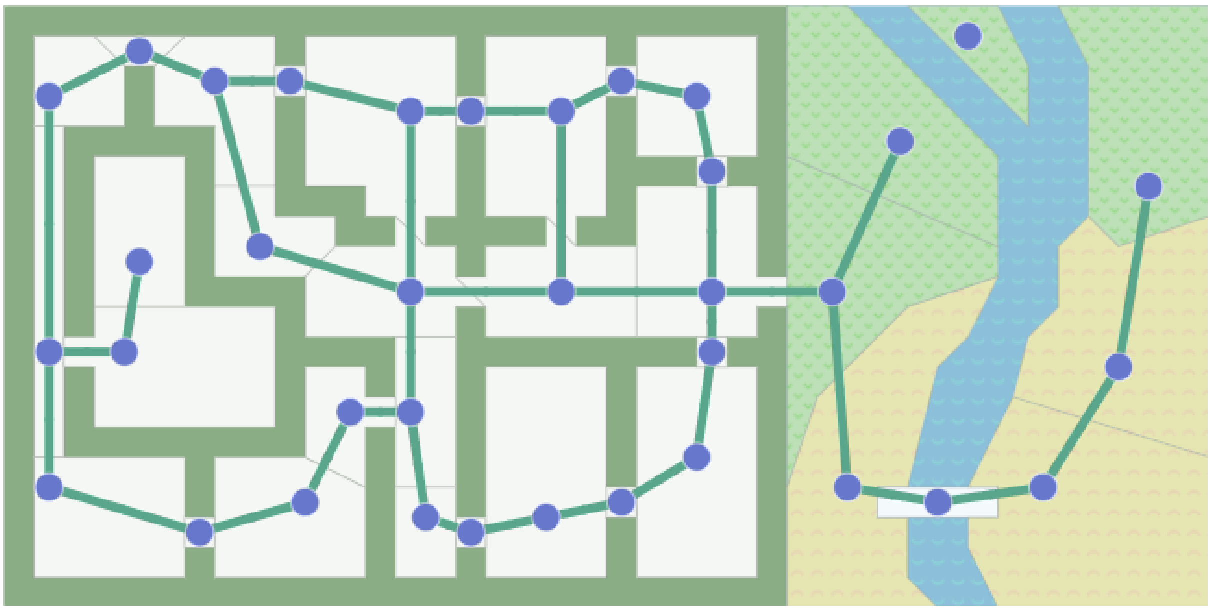
Dijkstra's Algorithm (also called Uniform Cost Search) - lets us prioritize which paths to explore. Instead of exploring all possible paths equally, it favors lower cost paths. We can assign lower costs to encourage moving on roads, higher costs to avoid forests, higher costs to discourage going near enemies, and more. When movement costs vary, we use this instead of Breadth First Search.

Complexity: **O((V+E) LogV) with min-priority queue**

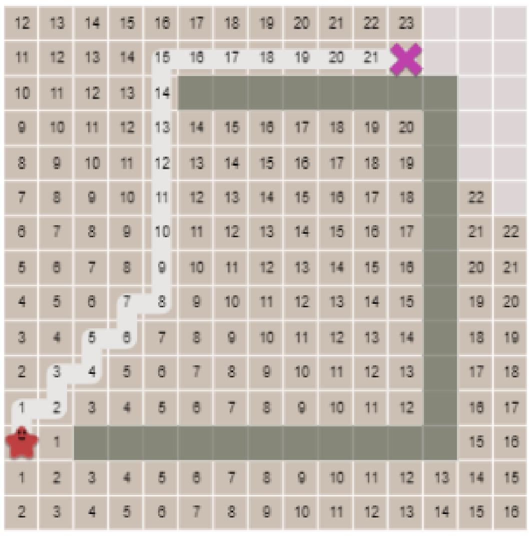
Floyd-Warshall Algorithm - Objectively finds the best path by finding the all-pairs shortest path. At every vertex we can start from we find the shortest path across the graph and see how long it takes to get to every other vertex. Each time we start over, we keep score of the total moves required for each vertex. The shortest average distance comes from the central vertex, which we can calculate with an adjacency matrix.

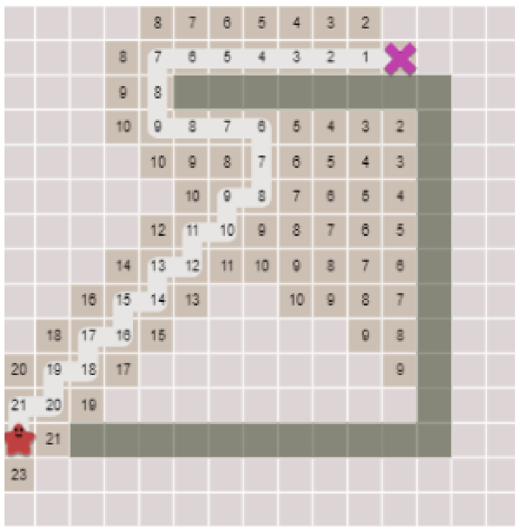
Complexity: **O(n3)**

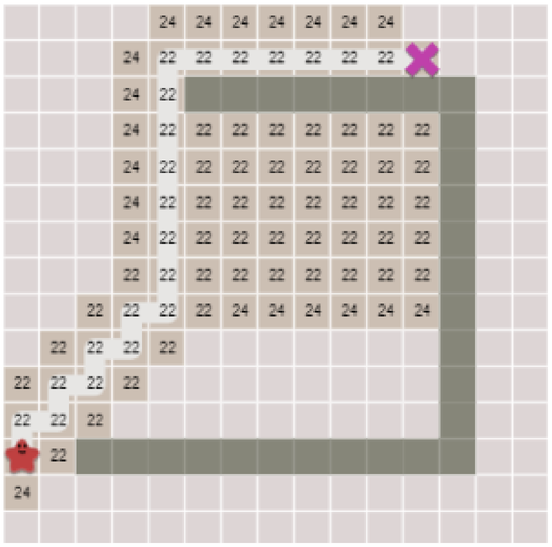
Output:



<u>Comparison Of Various Algorithms for the same Shortest-Path problem</u>


Djikstra's


Greedy BFS


A* Search

SCOPE OF GNN SYSTEM IN THIS PROBLEM

Graph Network (with attention read and write) can perform shortest path computations and generate the shortest path. The network performs this with 100% accuracy after minimal training.

Combination of different neural network components can make a network that readily learns a classical graph algorithm (such as the traditional ones mentioned above). Approximate methods faster than the classical algorithms were desired for computation of problems involving unseen graphs.

This project seeks to survey the Graph Neural Network system and its algorithm that works on many unseen graphs. The algorithm produces a network that will work on new, never previously seen graphs. It'll learn a graph algorithm to compute the shortest-path and also learn more complex graph functions from pairs of input questions and expected outputs.

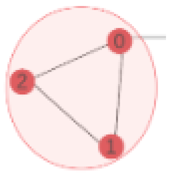
INPUT: < " How many stations are between station 5 and station 19 ", Rail Network >

OUTPUT: 7

The network is trained with ***Graph-Question-Answer*** tuples.

Each tuple contains a unique randomly generated graph, an English language question and the expected answer, split into training, validation and testing sets.

Example of a tuple:

<  , "How many stations are between station and station 19" , "7" >

The network is trained on the CLEVR Graph [which derives intuition from CLEVR (Compositional Language and Elementary Visual Reasoning) - a synthetic Visual Question Answering dataset, containing images of 3D-rendered objects; each image comes with a number of highly compositional questions that fall into different categories.]. It is a set of 10,000 graphs, each with an english language question and an answer to that question (generated as a YAML file).

There are eleven different question types; the network utilizes the question type:

How many stations are between Station x and Station y?

COMPONENTS INVOLVED IN THE GNN SYSTEM AND THEIR FUNCTIONALITY

The structure used is a Recurrent Neural Network (RNN). The network – which takes a question, performs multiple iterations of processing, and then produces an output – consists of an RNN cell, which is repeatedly executed, passing its results forwards.

First part of the system is building the input data pipeline –

→ The question - < How many stations… >

→ The input graph

→ The expected output answer

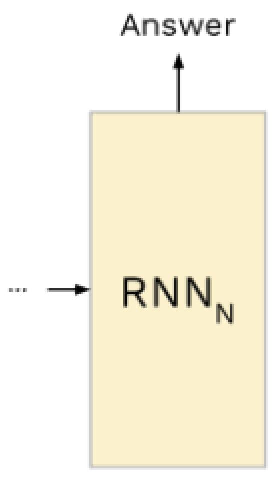
The RNN consists of a graph network which enables the system to compute functions of the graph's structure.

The final part of the system is the output cell which is essential to the network's success, which reads out from graph and takes previous steps' outputs.

The output cell has two parts:

i) Attention by index across the previous RNN cell outputs and the most recent read from graph network.

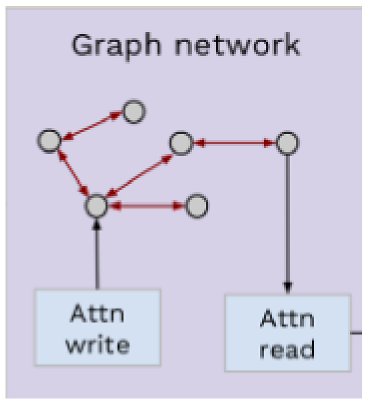
ii) A basic feed-forward network to transform the attention output into the cell's output.



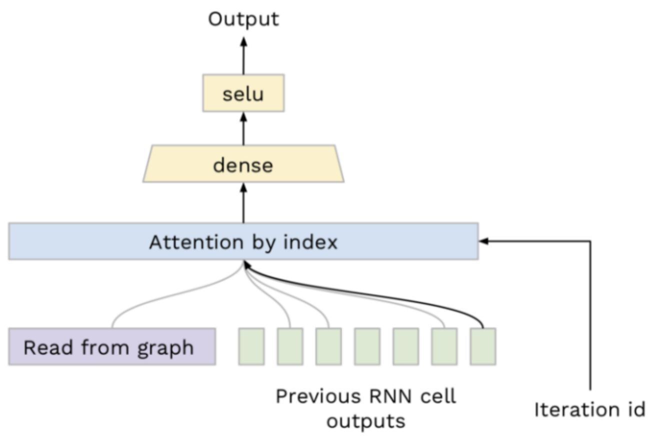
RNN Cell (Heart of the network)

<u>RNN Cell components</u>

i) Graph Network



ii) Output Cell



ALGORITHM

→ The RNN does four main things during an iteration (the system is run for 10 iterations).

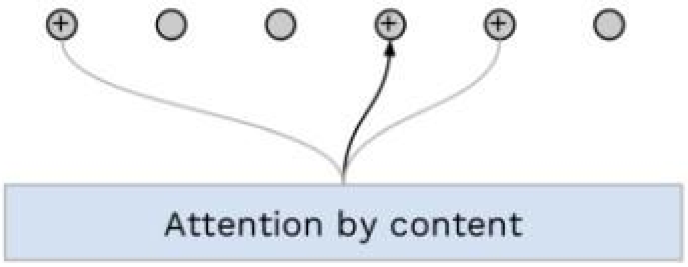
- Write data into a selected node’s state
- Propagate node-states along the edges in the graph
- Read data from a selected node’s state
- Take the read data, all previous RNN cell outputs, combine them, and produce an output for this RNN iteration

With these steps, the RNN is able to learn how to calculate shortest paths.

In the graph network each node n has a state vector S(n,t) (of width 4) at time t. In each iteration, the node states are propagated to the node’s neighbors. For shortest path calculations, a node state read and node state write is also required.

The initial states S(n,0) are zero valued vectors.

The node state write is a mechanism for the model to add a signal vector to the states of particular node(s) in the graph:



Algorithm of node-state read and write mechanisms:

Extract words from the question, to form the write query q_write

[*This will be used to select node states to add the write signal p to*]

1. The write query is generated using attention by index, which calculates which indices in the question words should be attended to (as a function of RNN iteration id r, which is a one-hot encoded vector) , then extracts them as a weighted sum.

$$\text{score} \leftarrow \sigma(Wr + b)$$

$$q_{write} \leftarrow \frac{1}{|Q|} \sum_{w \in Q} w \cdot \text{score}_w$$

2. A write signal p is computed by taking the RNN iteration id r and applying a dense layer with sigmoid activation.
3. The write signal and the write query are fed into an ‘attention by content’ layer to determine how the write signal will be added to the node states.
4. *Attention by content is simply the standard dot-product attention mechanism, where each item is compared with the the query by dot product, to produce a set of scores. The scores are then fed through softmax to become a distribution that sums to one.*

Here, scores are computed as the dot product of each node's state id with the write query.

5. The write signal is added to the node states in proportion to the scores.
6. The state is read from the graph, the same way it was written.
7. A read query is calculated from the input question words using attention by index.
8. The read query is then, used to extract a value from the node states using attention by content.
9. Scores are computed as the dot product of each node's state id with the read query.
10. The final read out value is computed using the weighted sum of the node states.

$$\frac{1}{|N|}\sum_{n\in N} S_n \cdot \mathrm{score}_w$$

The final part of the system is the Output Cell, which is necessary for the system's success.

Output cell combines the outputs from earlier iterations with the current graph network output, allowing the cell to repeatedly combine previous outputs.

This also helps the network to easily look back to the output of an earlier iteration, regardless of total number of RNN iterations.

ANALYSIS OF RESULTS

The hyperparameters of the network were determined experimentally.

Eg: Learning Rate - Learning Rate Finder Protocol

Node state size, number of graph read/write heads and number of RNN iterations - Grid Search

The network achieves 100% test accuracy after 9k training cycles (only 2 mins on a MacBook Pro CPU), thus showing that the network shows strong inductive bias towards solving this problem.

Complexity : ~ O(|E|) , similar to Djikstra's complexity which is O(|E| |N|log|N|), where E: edges, N: nodes

Compared to the classical approach for shortest path finding - Djikstra's - the algorithm is less efficient since it requires initial training, whereas Djikstra's does not, and requires more operations (although the runtime can be the same by using GPUs for the parallel matrix operations).

Although less efficient, the system has the benefit of being able to learn different functions depending on the training examples.

RESULTS

Examples of work in this area include Neural networks for routing communication traffic (1988), A Neural Network for Shortest Path Computation (2000) and Neural Network for Optimization of Routing in Communication Networks (2006).

A recent, groundbreaking approach to the problem was DeepMind's Differentiable neural computers (PDF), which computed shortest paths across the London Tube map.

Thus, Graph Neural Networks can be utilised for effectively solving a variety of Optimization problems, and could be used in the future for even more applications.

REFERENCES

[1] ”*Finding Shortest Path with Graph Networks - Octavian.ai*” *https://github.com/Octavian-ai/shortest-path.*

[2] Fatemeh Salehi Rizi, Joerg Schloetterer,Michael Granitzer, *”Shortest Path Distance Approximation using Deep Learning Techniques”, 2020.*

[3] Kevin Osanlou, Christophe Guettier, Andrei Bursuc, Tristan Cazenave, Eric Jacopin,*“Constrained shortest path search with graph convolutional neural networks”,2021.*

[4] Krzysztof Rusek, Jose Suarez-Varela, Paul Almasan, Pere Barlet-Ros and Albert Cabellos-Aparicio, *“RouteNet: Leveraging Graph Neural Networks for network modeling and optimization in SDN”,2020.*