Applying the Neural Bellman-Ford Model to the Single Source Shortest Path Problem

Spyridon Drakakis[a] and Constantine Kotropoulos[b]
Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece

Keywords: Message-Passing Neural Networks, Graph Neural Networks, Bellman-Ford Algorithm, Single-Source Shortest Paths.

Abstract: The Single Source Shortest Path problem aims to compute the shortest paths from a source node to all other nodes on a graph. It is solved using deterministic algorithms such as the Bellman-Ford, Dijkstra's, and A* algorithms. This paper addresses the shortest path problem using a Message-Passing Neural Network model, the Neural Bellman Ford network, which is modified to conduct Predecessor Prediction. It provides a roadmap for developing models to calculate true optimal paths based on user preferences. Experimental results on real-world maps produced by the Open Street Map package show the ability of a Graph Neural Network to imitate the Bellman-Ford algorithm and solve the Single-Source Shortest Path problem.

1 INTRODUCTION

The Optimal Path problem (Martins et al., 1999) is the problem of calculating the optimal path from one point to another. The vagueness in the definition of optimality is purposeful since its generality allows for complex and arbitrary criteria to be encompassed in the definition. Instances of this general problem are the Single-Source Single-Destination and the Single-Source Shortest Path (SSSP) problem. In this paper, the SSSP problem is considered.

The SSSP problem is the problem of calculating the shortest paths from a source node $u \in \mathcal{V}$ to all other nodes, given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. There are plenty of classical algorithms that solve this problem, such as Breadth-First Search (BFS), Dijkstra's Algorithm (Dijkstra, 1959), Bellman-Ford Algorithm (Bellman, 1958), A* (Hart et al., 1968), efficiently. However, these algorithms cannot model complex criteria and, except A*, cannot model any heuristic information at all. Thus, they are restricted to optimizing a limited amount of criteria, or one criterion, at a time and cannot solve the general Optimal Path problem.

The prominence of Neural Networks and their capacity to optimize complex cost functions make them ideal as a vehicle to regain the vagueness and generality in the definition of optimality caused by solving a particular instance of the Optimal Path problem, namely the SSSP problem. Specifically, we will consider Message-Passing Neural Networks (MPNNs), a type of Graph Neural Network (GNN) suited for reasoning and making predictions on graphs.

MPNNs require that the edges and nodes of the graph have defined representations. The classical algorithms use the weights of the edges as a representation of the edges and the distance from the source node as a representation of the nodes, both scalar. In (Zhu et al., 2021), a very sound theoretical foundation was set for path-oriented node representation algorithms, encapsulating several classical algorithms. An analogous process can be used to define an MPNN. Also, we will be taking inspiration from the Bellman-Ford algorithm and, first, show how it can be parameterized by a message-passing process and, second, define the SSSP problem in a way that an MPNN can solve. The choice of the Bellman-Ford algorithm lies in its inclination for parallelization and its easy implementation as a message-passing process.

In this paper, we present a novel approach regarding the analysis of MPNNs, according to which we address failures caused by restraints (such as a dead end on a directed graph) and provide proper and efficient recovery solutions. We outline critical limitations of the model and ways to overcome them, provide visualizations and explanations of instances of failure, and outline solutions based on the observations.

[a] https://orcid.org/0009-0001-1253-9498
[b] https://orcid.org/0000-0001-9939-7930

Drakakis, S. and Kotropoulos, C.
Applying the Neural Bellman-Ford Model to the Single Source Shortest Path Problem.
DOI: 10.5220/0012425800003654
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 13th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2024)*, pages 386-393
ISBN: 978-989-758-684-2; ISSN: 2184-4313
Proceedings Copyright © 2024 by SCITEPRESS – Science and Technology Publications, Lda.

The outline of the paper is as follows. In Section 2, some related works on GNNs and their use in solving the SSSP problem are presented. In Section 3, the model derivation is explained. We describe the Bellman-Ford algorithm and MPNNs and discuss how they are connected. Then, we explain how the Bellman-Ford algorithm can be generalized and parameterized by an MPNN, and we subsequently define such an MPNN for the SSSP problem. In Section 4, experimental results are presented. We provide an analysis of the operation of the model on trivial and real-world maps, outline how to overcome the limitations, and analyze instances of failure, proposing suitable recovery methods from failures. In Section 5, concluding remarks are given.

2 RELATED WORK

This section reviews related works on GNNs and their use in solving path problems.

(Zhu et al., 2021) showed that many classical algorithms can be encapsulated by a general path formulation and laid the theoretical foundation behind the Generalized Bellman-Ford algorithm (Baras and Theodorakopoulos, 2010). They also developed the Neural Bellman-Ford model to solve the link prediction problem, which is essentially the model used in this paper, with adjustments to apply it to the SSSP problem. They used pair representations of nodes produced by examining the local subgraph between two nodes to predict links, achieving 95% average precision on the Cora dataset, 93% on the Citeseer dataset, and 98% on the Pubmed dataset.

(Veličković et al., 2020) developed a neural network model that could execute several classical algorithms on graphs at once. Specifically, using an MPNN for best results, they solved the neural execution task, executing BFS, the Bellman-Ford algorithm, and Prim's algorithm (Prim, 1957) all at once. They did this using the encoder-process-decoder paradigm (Battaglia et al., 2018), according to which 3 neural networks, an encoder, a process, and a decoder network are employed to transform the intermediate inputs and share them with the algorithms being learned. Also, a termination network is needed to decide the termination condition of the algorithms. Related to this paper is the Predecessor Prediction they conducted as part of their tests, using the BFS algorithm for reachability and the Bellman-Ford algorithm simultaneously to predict predecessors of nodes of the shortest paths, achieving accuracies of 97.13%, 94.71%, 90.91%, 83.08%, 77.53%, 74.90% for graph sizes of 20, 50, 100, 500, 1000, 1500 nodes, respectively.

(Wu et al., 2020) used a mixture of GNN models to learn a hierarchical representation of road networks. The hierarchy is *functional zones* $\rightarrow$ *structural zones* $\rightarrow$ *road segments*, where *functional zones* are sets of *structural regions*, indicating some kind of traffic functionality, *structural regions* are sets of spatially connected *road segments* and *road segments* are uniform sections of road. They constructed representations of these entities by using embeddings for *road segments*, spectral clustering and graph attention networks for *structural regions*, and another graph attention network for *functional zones*. Focusing on the route planning task, which is the most similar to our case, given an actual path p and a produced path p', they achieved F1 scores of 0.329, 0.357, 0.301 and edit distances of 7.851, 7.361, 8.138 for road networks of Beijing, Chengdu, and Xi'an, respectively.

In the same vein, (Huang et al., 2021) used another hierarchical embedding approach, partitioning the road network into subgraphs (trees) with a minimum-cut approximation, calculating embeddings for each one. Then, the local embeddings (representation of the nodes) are learned by minimizing the L_1 distance. Some nodes are designated as landmarks, and training samples are chosen uniformly from the sets of closest vertices to each landmark, ensuring that the training sample is diverse. Then, fine-tuning is performed using more training samples from those vertices that produce the highest approximation errors. They achieved mean error rates (against actual shortest paths) of 0.60%, 0.63%, and 0.48% for road networks of Beijing, Florida, and Western-USA regions, respectively.

3 METHODOLOGY

The process by which the MPNN is derived is explained in detail. The process consists of four steps:

- Understanding the Bellman-Ford algorithm.
- Using a Message-Passing process to simulate the Bellman-Ford algorithm.
- Incorporating Neural Networks as part of the message-passing.
- Formulating the SSSP problem as a classification problem.

3.1 Bellman-Ford Algorithm

The Bellman-Ford algorithm (Bellman, 1958) is a dynamic programming algorithm calculating the shortest paths from a source node u to all other nodes v_i, for $i = 1,2,...,|\mathcal{V}|$. It does so by storing node representations $h^{(t)}(u,v_i)$ for each node, which is the current shortest distance to the source node, and updating them every time there is an alternative path from u to v that is shorter than the known path. That is, for every node v_i, and an incoming edge e_{xv_i} with edge representation its weight $w(x,v_i)$, the update (also called relaxation) is (Li et al., 2022):

$$h^{(t)}(u,v_i) = \min\left(h^{(t-1)}(u,x) + w(x,v_i), h^{(t-1)}(u,v_i)\right) \quad (1)$$

This process is done for every node, i.e., $|V|-1$ times per iteration. Therefore, relaxation is conducted $|V|-1$ times on the edges. In the beginning, the representation of the nodes $h^{(0)}(u,v_i)$ is initialized by:

$$h^{(0)}(u,v_i) = \begin{cases} 0 & \text{if } u = v_i \\ \infty & \text{otherwise.} \end{cases} \quad (2)$$

The key constituents of the algorithm are easily detected. Firstly, there is an initialization based on equation (2), also called the boundary condition. Then, every node receives the previous representation of its neighbors and updates its representation if it is less than its own. In other words, it collects the representations of its neighbors and its own and keeps the minimum. This alternative wording of the algorithm's inner workings provides the recipe for an equivalent message-passing operation.

3.2 Message-Passing Neural Networks

MPNNs, as described in (Gilmer et al., 2017), are a general class of GNNs since many GNN architectures, like Graph Convolutional Networks (Kipf and Welling, 2017), can be described in terms of an MPNN. The basic idea of an MPNN is that information about the representation of a node is linked to the node representations of its neighbors. In other words, a better representation of the node can be achieved by aggregating information from its neighborhood.

The message-passing operation conducted in an MPNN consists of three steps:

1. Every node computes a *message* for every one of its neighbors. This message is a function of its own representation, that of the neighbor, and the representation of the edge linking the two.
2. The messages are sent, and every node *aggregates* the messages it receives using a permutation invariant function, meaning the order in which the messages are received does not matter.
3. Every node *updates* its representation as a function of its current representation and the aggregated messages.

The operations correspond to three different functions, *message*, *aggregate*, *update*. A key feature of an MPNN is the type of functions it employs. Fundamental message functions as relational operators on graph embeddings are natural summation as in (Bordes et al., 2013), multiplication as in (Yang et al., 2015). Some important aggregators are natural summation, the mean operator, maximum, and minimum.

MPNNs consist of layers, like regular Neural Networks. In each layer, a message-passing operation is conducted, and information flows through the model in a similar fashion to feed-forward neural networks. The output of one layer, followed by a non-linear activation, is the input of the next.

If we focus on a single node v_i, every time a message-passing operation is conducted, i.e., the input flows through the layers of the model, the node receives information from nodes k steps away, where k is the number of message-passing operations conducted.

The message-passing operation can easily be shown to be equivalent to the Bellman-Ford iteration (Gallager, 1982). Focusing again on a single node v_i, we see that the node updates its distance to the source node by examining the difference in its distance to the source node if it uses a different incoming edge. As stated before, this process can alternatively be defined as gathering messages from every neighboring node (covering all incoming edges) and updating its node representation by taking the minimum of all the messages. Finally, to include the node itself in the update operation, we attach a self-loop to each node, allowing it to send a message to itself to compare to the others. This results in a different but similar update rule:

$$h^{(t)}(u,v_i) = \min\left(h^{(t-1)}(u,x) + w(x,v_i), h^{(0)}(u,v_i)\right) \quad (3)$$

where $w(x,v_i) = 0$ for $x = v_i$. By conducting multiple message-passing operations, we can fully simulate the Bellman-Ford algorithm using message-passing operations.

3.3 Generalizing the Bellman-Ford Algorithm

One can observe that the representation of a node $h^{(t)}(u,v_i)$ is the representation of the shortest path $h(P)$ from u to v_i, where $P = e_1 e_2 ... e_{|P|}$, i.e.,

$$h(u,v_i) = h(P) \quad (4)$$

$$h(P) = w(e_1) + w(e_2) + \ldots + w(e_{|P|}) \quad (5)$$

where $||$ stands for set cardinality. Generally, finding the shortest path between two nodes can be formulated as a problem of enumerating all the possible paths between the two nodes and keeping the one with the minimum cost. That is (Zhu et al., 2021),

$$h(P=(e_1,e_2,\ldots,e_{|P|})) = w(e_1)+w(e_2)+ \\ + w(e_{|P|}) \tag{6}$$

$$h(u,v_i) = \min\left(h(P_1),h(P_2),\ldots,h(P_{|P_{u,v_i}|})\right) \tag{7}$$

where P_{u,v_i} is the set of all possible paths between u and v_i. If we allow general node and edge representations that are no longer scalar and generalized operations, $\otimes, \oplus$ replacing natural summation and the min operator, equations (6) and (7) become (Zhu et al., 2021):

$$\mathbf{h}(P=(e_1,e_2,\ldots,e_{|P|})) = \mathbf{w}(e_1)\otimes\mathbf{w}(e_2)\otimes\ldots \\ \mathbf{w}(e_{|P|}) \triangleq \bigotimes_{i=1}^{|P|}\mathbf{w}(e_i) \tag{8}$$

$$\mathbf{h}(u,v) = \mathbf{h}(P_1)\oplus\mathbf{h}(P_2)\oplus\ldots\oplus\mathbf{h}(P_{|P_{u,v}|})| \quad P_i \in P_{u,v} \\ \triangleq \bigoplus_{P\in P_{u,v}} \mathbf{h}(P) \tag{9}$$

This formulation is very computationally expensive as it requires that all the possible paths are calculated, yielding exponential complexity. If we now substitute our new operators, node, and edge representations in equations (3) and (2), we get the generalized Bellman-Ford algorithm. The generalized operators satisfy a semi-ring system with the neutral element for summation $\textcircled{0}$ and the neutral element for multiplication $\textcircled{1}$. Additionally, we consider a homogeneous graph to be a Knowledge Graph ($\mathcal{KG}$) $\mathcal{G}=(\mathcal{V},\mathcal{R},\mathcal{E})$, where $\mathcal{R}$ is the set of relations on the graph, with one relation. Thus, we have the following algorithm (Li et al., 2022):

$$\mathbf{h}_q^{(0)}(u,v) \longleftarrow \mathbb{1}_q(u=v) \tag{10}$$

$$\mathbf{h}_q^{(t)}(u,v) \longleftarrow \left(\bigoplus_{(x,r,v)\in\mathcal{E}(v)} \mathbf{h}_q^{(t-1)}(u,x)\otimes\mathbf{w}(x,r,v)\right) \\ \oplus\,\mathbf{h}_q^{(0)}(u,v), \quad t\geq 1 \tag{11}$$

where q is the query relation and $\mathbb{1}_q(u=v)$ is an indicator function that outputs $\textcircled{1}_q$, if $u=v$ and $\textcircled{0}_q$ otherwise. $\textcircled{1}_q$ and $\textcircled{0}_q$ are now neutral elements for summation and multiplication with respect to the query relation q.

3.4 Incorporating Neural Networks

To achieve generalization capacity, neural networks must be incorporated into the model. For this purpose, we will define three neural functions: *message*, *aggregate*, and *indicator*.

The indicator function serves the purpose of a non-trivial boundary condition. So, instead of initializing node and edge representations according to equation (2), we use an embedding (Yang et al., 2015), (Kazemi and Poole, 2018) for a shared representation for the edges $\mathbf{w}(e)$ and initialize the source node with this representation.

The message function serves the purpose of the $\otimes$ operator. It will be implemented as a linear combination of the representation of the node x sending the message and the edge representation of the edge connecting x and v, where v is the node receiving the message. The aggregate function is going to be implemented as the min operator. After the aggregation function, a non-linear activation is applied to the node representations.

The consequence of these changes is that we have incorporated learnable parameters and can generalize and classify nodes. Equation (11) becomes (Li et al., 2022).

$$\mathbf{h}_q^{(t)}(u,v) \longleftarrow \min\Bigg[\min_{(x,r,v)\in\mathcal{E}(v)}\Big(\mathbf{h}_q^{t-1}(u,x)+ \\ + w_{x,v}\,\mathbf{w}(x,r,v)\Big),\mathbf{h}_q^{(0)}(u,v)\Bigg], \quad t\geq 1 \tag{12}$$

where $w_{x,v}$ is the weight of the edge $e(x,v)$. A Multilayer Perceptron is also included at the end to classify the nodes based on the node representations.

3.5 SSSP Problem as a Classification Problem

The SSSP problem can be modeled as a classification problem. Instead of classifying nodes or edges as optimal or non-optimal, we can model the problem as one of Predecessor Prediction, taking inspiration from the original Bellman-Ford algorithm. Thus, Predecessor Prediction is a multi-class classification problem with a number of classes $C=|\mathcal{V}|$. We classify each node in one of these classes, corresponding to its predecessor node. We have a label for every node that can be calculated using one of the classical algorithms, Bellman-Ford or Dijkstra. The loss function of such a problem is the Cross-Entropy loss.

4 EXPERIMENTAL RESULTS

The models were trained with the Stochastic Gradient Descent (SGD) algorithm using the Adaptive Moment Estimation (ADAM) optimizer. The learning rate used is $5 \cdot 10^{-3}$ and a batch size 16. The other hyperparameters, such as model architecture, message-passing iterations, and the number of epochs, vary and are summarized in Table 1.

Experiments were conducted on 4 small graphs produced by the Open Street Map package. The graphs are real-world graphs of 3 locations. The first two stem from the same region, Lagadas [1], and the remaining three are different regions of the city of Thessaloniki, one of the center of Thessaloniki [2], one of the region around the Aristotle University of Thessaloniki [3] and one of the region of Evangelistria [4]. Before presenting the results, analyzing the model on trivial graphs and understanding its weaknesses is useful.

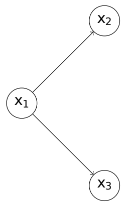

Figure 1: A simple graph with three nodes.

Consider the graph in Fig. 1. In this trivial graph, there are three nodes and three edges. If we split the nodes, as is common practice, in three sets (training, validation, test), the problem arising is presented in Fig. 2. Assuming that the training set includes the nodes x_2,x_3 and the test set includes the node x_1, the model can only learn the trivial rules `pred`(x_2) = x_2 and `pred`(x_3) = x_3. If it attempts to predict with source node x_1, it is bound to fail since it has learned that given a source node x_i, its predecessor is the node itself `pred`$(x_i) = x_i$. This result indicates that the choice of the training, validation, and test sets is crucial for the model's success. As a proof of concept, if the training set equals the validation and test sets and is equal to the whole dataset, then the training is concluded successfully, and the model can predict the predecessors for all the nodes correctly. This, of course, is an instance of overfitting, generally intolerable (Hawkins, 2004). However, it provides useful insight into the workings of the model.

Moving on to the next graph, presented in Fig. 3, we have a very small graph of the region of Lagadas

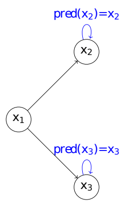

Figure 2: The rules learned by the model after training.

with 21 nodes. The results in this graph prove that the model can fully imitate the Bellman-Ford algorithm and present a problem with generalization.

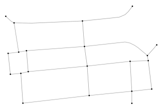
Figure 3: Small graph of the region of Lagadas.

First, the model achieves 99% accuracy when the training set is equal to the test set. This fact proves its ability to imitate the Bellman-Ford algorithm deterministically. The problem presented in generalization is that the model can get stuck into infinite loops, either cycles or constant switching between 2 nodes without restrictions. This phenomenon is explained in Fig. 4, where the orange line is the true shortest path, the blue line is the walk produced by the model (note that it is not a path since every node is not unique), and the pink line is the intersection of the two. The model is stuck in an infinite loop between the two nodes indicated by the black arrow.

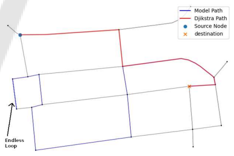

Figure 4: An instance of failure on the graph of Lagadas.

To understand this phenomenon better and to present a solution, consider the graph of Fig. 5. It is a simple graph with a dead end at the node x_7. To be precise, we can observe that the dead end starts at node x_5 since there is no prospect of recovery after the model chooses this node as a predecessor node. The problem is that the following scenario leads to a failure when reaching node x_4 if the model makes a wrong prediction, i.e., node x_5, since the model cannot escape its predicament without intervention.

A recovery algorithm will be deployed to reconstruct the shortest path to combat this weakness. The intuition of the algorithm is presented in Fig.6. When the reconstruction algorithm reaches node x_7, the recovery algorithm will back-track until it reaches the nearest node for which there is more than one choice

[1]https://www.openstreetmap.org/export#map=16/40.7497/23.0724
[2]https://www.openstreetmap.org/export#map=16/40.6362/22.9477
[3]https://www.openstreetmap.org/export#map=16/40.6338/22.9543
[4]https://www.openstreetmap.org/export#map=16/40.6344/22.9618

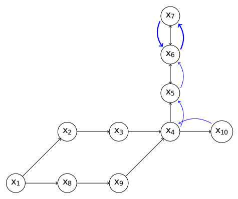

Figure 5: A simple graph that contains a dead end.

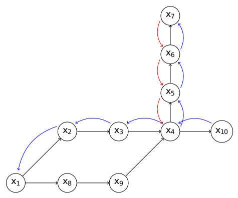

Figure 6: The intuition behind the recovery algorithm.

for a predecessor node. This last point is essential because taking as few steps as possible is desirable, just enough to steer the algorithm away from the dead end. However, if we only took one step back to node x_6, the reconstruction algorithm would end up at node x_7. If we reach node x_4, the algorithm can recover if we additionally forbid it to choose a node already in its path. Thus, the next choice for a predecessor node would be node x_2, escaping the dead end. The algorithm is presented in Algorithm 1.

Algorithm 1: The recovery algorithm.

```
Data: path, list of nodes in current path
Result: Alternative choice $v$ that does not lead
        to a dead end
;
while $len(pred[v]) = 1$ do
    path.pop()
    $v$ = path[len(path)-1]
end
$v$ = prev[$v$][$k$], $k$ is the index of the
second largest element of pred[$v$]
```

This intervention naturally affects predictions. During testing, the algorithm will not be deployed to maintain the integrity of the results with respect to accuracy. This process, however, is useful to guarantee valid paths when the model fails on its own. This algorithm has computational complexity $O(|\mathcal{V}|)$ since, worst-case, every node is going to be visited no more than two times, one time when the algorithm reaches the dead end, plus one time while back-tracking.

Inserting the recovery algorithm into our arsenal, the analysis proceeds to the graph presented in Fig. 7. This is a graph of the center of Thessaloniki. This graph provides insight into how the model suffers in the range it can cover. As explained in Section 3.2, an MPNN model covers the k-neighborhood of a node,

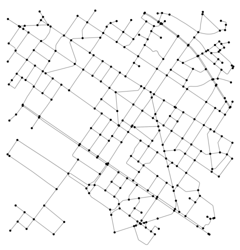

Figure 7: The graph of the center of Thessaloniki.

where k is the number of layers of the network. This is not a problem with graphs of high node degree (e.g., knowledge graphs), but it is a problem when dealing with low node degree graphs, like road networks. To understand this fact, consider that the paths from one node to another in a road network can be quite large so that nodes of interest escape the k-neighborhood of a source node. Consequently, a message from the source node is not guaranteed to reach a node, which presents a problem when constructing a path of great length. One solution is to deepen the model by adding more layers. However, this solution presents more computational complexity and more learnable parameters, leading to difficulties in training, and is not guaranteed to work at all (Sun et al., 2015).

The solution is comprised of two parts. First, we iteratively conduct the message-passing operation with the same model (i.e., a 6-layer model). This guarantees that a message from the source node can reach every other node. The choice of the number of iterations depends on each graph, and it has to do with the maximum distance (in steps) to the most remote node in the graph. In the worst case, the path from the source node to the most remote node is of length $|\mathcal{V}|$ if all nodes are consequentially sorted. However, in practice, the paths that need to be covered are much shorter. Second, suppose the unaltered distances between the nodes are used for the weights of the graph edges $w(e)$ (e.g., in meters). In this case, numerical instability is introduced into the model since the messages sent and the neural network model weights are too large, leading to failure in training. To combat this phenomenon, a regularization of the weights is called for. The aim is to scale them so that iterative additions maintain the messages and the weights of the neural network small. The regularization chosen is the min-max regularization, given by

$$w' = \frac{w - \min(\mathcal{A})}{\max(\mathcal{A}) - \min(\mathcal{A})}(\mathtt{new\_max}(\mathcal{A}) - \mathtt{new\_min}(\mathcal{A})) + \mathtt{new\_min}(\mathcal{A}) \qquad (13)$$

where min and max are the minimum and maximum elements of the set $\mathcal{A}$, and new_min, new_max are the desirable new bounds. The result is that the elements of set $\mathcal{A}$ are in $[\texttt{new\_min}, \texttt{new\_max}]$, and the relative distances of the elements are maintained. The problem with this regularization is that outliers typically greatly affect the quality of the regularization. In our case, the datasets do not contain noise, so this regularization is very powerful. We define the new interval as $[0.2, 1]$ since a weight of 0 has no meaning on a graph with no self-loops. After this regularization, the extension of the range of the message-passing process is possible, and the result is presented in Fig. 8.

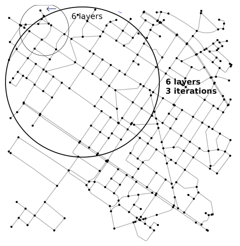

Figure 8: The expansion of the range of the message-passing operation.

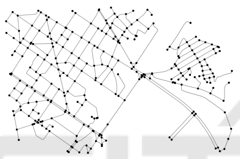
Figure 9: A graph of the region around the Aristotle University of Thessaloniki.

The experimental results are summarized in Table 1. The accuracy refers to the mean accuracy achieved when predicting predecessors, given a source node. Another metric given is the mean distance of the paths produced by the model and the true shortest paths. In Table 1, the mean distance from the shortest path is given with the deployment of the recovery algorithm and without. As illustrated shortly, the metrics presented do not give a holistic picture of the model's efficiency.

The Evangelistria graph, Fig. 11 is a graph of a region in Thessaloniki, and its analysis is comparable to the small graph of Lagadas presented. The Aristotle University of Thessaloniki (AUTH) and Big Lagadas graphs are comparable to the graph of the center of Thessaloniki. They are shown in Fig. 9 and Fig. 12.

Table 1: Results from experimental results.

Graph	# layers	# iterations	Accuracy	Mean Distance (with/ without recovery)	# epochs
Small Lagadas	4	3	90%	3m/218m	160
Big Lagadas	2	4	91%	3m/1370m	180
Evangelistria	2	4	91%	6.8m/1210m	360
Center of Thessaloniki	4	30	75%	2643m/11982m	20
AUTH	4	20	83%	1652m/15629m	20

One issue with the metrics given is that they do not take into account the damage caused by instances of

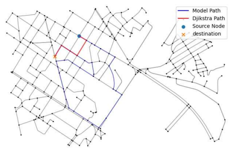

Figure 10: The rules learned by the model after training.

failure, because the overwhelming number of correct predictions smooths accuracy, and the overwhelming number of correct paths smooths the mean distance. The metrics give us an idea concerning the general quality of the model. However, they do not provide us with insight regarding instances of failure. Such an instance is presented in Fig. 10, the graph of AUTH. This is an instance of failure, but with the use of the recovery algorithm, the model can return a valid path severely longer than the optimal path, 1663m to be exact. Using the recovery algorithm, we can guarantee valid paths that are naturally worse than the optimal ones. This algorithm was a product of analyzing the model's inner workings and instances of failure while trying to find a cure. Many failures can be prevented, and the results can be improved.

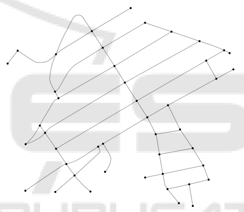
Figure 11: A graph of the region of Evangelistria.

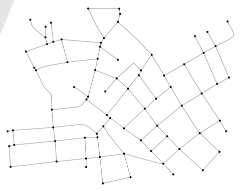
Figure 12: A bigger graph of the region of Lagadas.

5 CONCLUSIONS

We have used the Neural Bellman-Ford Network to solve the Single-Source Shortest path problem. The problem has been defined as a Predecessor Prediction task, taking inspiration from the original Bellman-Ford algorithm. The Predecessor Prediction task has been modeled as a node classification task suited to be solved by a Message-Passing neural network. We have examined the weaknesses of a Message-Passing neural Network model when operating on a road network. We have proposed a solution. This paper can be considered a stepping stone to solving the general Optimal Path problem, which can provide great benefits to professionals and others if efficiently and sufficiently solved. Another aim of the paper has been to motivate the integration of classical algorithms with the latest state-of-the-art methods and the solution of fundamental problems using those methods.

ACKNOWLEDGEMENTS

This research was carried out as part of the project "Optimal Path Recommendation with Multi Criteria" (Project code: KMP6-0078997) under the framework of the Action "Investment Plans of Innovation" of the Operational Program "Central Macedonia 2014-2020", that is co-funded by the European Regional Development Fund and Greece.

REFERENCES

Baras, J. S. and Theodorakopoulos, G. (2010). Path problems in networks. *Synthesis Lectures on Communication Networks*, 3:1–77.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Bellman, R. (1958). On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Gallager, R. G. (1982). Distributed minimum hop algorithms. Technical Report LIDS-P 1175, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hawkins, D. M. (2004). The problem of overfitting. *Journal of Chemical Information and Computer Sciences*, 44(1):1–12. PMID: 14741005.

Huang, S., Wang, Y., Zhao, T., and Li, G. (2021). A learning-based method for computing shortest path distances on road networks. In *Proceedings of the 2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 360–371. IEEE.

Kazemi, S. M. and Poole, D. (2018). Simpl embedding for link prediction in knowledge graphs. *arXiv preprint arXiv:1802.04868*.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Li, J., Zhuo, L., Lian, X., Pan, S., and Xu, L. (2022). DPB-NBFnet: Using neural Bellman-Ford networks to predict dna-protein binding. *Frontiers in Pharmacology*, 13.

Martins, E., Pascoal, M., Rasteiro, D., and Santos, J. (1999). The optimal path problem. *Investigacao Operacional*, 19:43–60.

Prim, R. C. (1957). Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401.

Sun, S., Chen, W., Wang, L., Liu, X., and Liu, T.-Y. (2015). On the depth of deep neural networks: A theoretical view. *arXiv preprint arXiv:1506.05232*.

Veličković, P., Ying, R., Padovano, M., Hadsell, R., and Blundell, C. (2020). Neural execution of graph algorithms. *arXiv preprint arXiv:1910.10593*.

Wu, N., Zhao, X. W., Wang, J., and Pan, D. (2020). Learning effective road network representation with hierarchical graph neural networks. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 6–14.

Yang, B., Wen-tau Yih, He, X., Gao, J., and Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*.

Zhu, Z., Zhang, Z., Xhonneux, L.-P., and Tang, J. (2021). Neural Bellman-Ford networks: A general graph neural network framework for link prediction. In *Advances in Neural Information Processing Systems*, volume 34. MIT-Press.