MASTER DEGREE THESIS

MASTER'S DEGREE IN COMMUNICATIONS AND COMPUTER NETWORKS ENGINEERING

Scalable routing and wavelength assignment in large optical networks

POLITECNICO DI TORINO

Author:
Neman ABDOLI
248228

Supervisor:
Dr. Andrea BIANCO

Co-supervisor:
Dr. Cristina ROTTONDI

JULY, 2020

A thesis submitted in fulfillment of the requirements for the

Master's degree in Communications and Computer Networks Engineering

Written in

Telecommunication Networks Group
Department of Electronics and Telecommunications
Politecnico di Torino

Abstract

Scalable routing and wavelength assignment in large optical networks

Recent advances in Graph Neural Networks (GNN) have shown a dramatic improvement in the solution of computer networks problems. GNN seems promising to solve many relevant network optimization problems (e.g., routing) in self-driving software-defined networks. However, most state-of-the-art GNN-based networking techniques fail to generalize, which means that they perform well in network topologies seen during training, but not over large topologies. The reason behind this important limitation is that existing GNN networking solutions use standard graph neural networks that are not suited to learn large graph-structured information in routing purposes. In this thesis, we propose to use Massage Passing Neural Networks (MPNN).

MPNN is tailored to learn and model information structured as graphs and as a result, our model is able to generalize over arbitrary topologies and variable traffic intensity. To showcase its generalization capabilities, we evaluate it on a SDN-based Optical Transport Network (OTN) scenario, where traffic demands need to be allocated efficiently. Our results show that our model is able to achieve outstanding performance in large topologies never seen during training.

KEYWORDS: Machine learning, Graph neural network, shortest path problem, routing and wavelength assignment, Graph embedding

Acknowledgements

I would like to express my gratitude to those who helped to this thesis and support my master research. My deepest appreciate goes to my supervisors, Prof. Andrea Bianco and Dr. Cristina Rottondi for their support and motivation. I have learned a lot about the correct way of research and you taught me how to look deeper in subjects and to be critical. I am thankful for your helpful comments and advices during my work.

Last but not least, I would like to express my deepest gratitude to the most valuable person in my life. My wife, Youkabed, who have always been there to support me in my life. This work is dedicated to my family.

List of Contents

Abstract I
Acknowledgements III
Introduction 1
1.1 Motivation 1
1.2 Thesis outline 2
Related works 4
2.1 ML-based shortest path computation 4
2.1.1 Auto Computed Neural Network (ACNN): 4
2.1.2 Stochastic Shortest Path-based Q-learning (SSPQ) 6
2.2 ML techniques in routing and wavelength assignment (RWA) 8
2.2.1 ML-Based Quality of Service Routing 8
Supervised Machine Learning-Based Routing and Wavelength Assignment 9
2.3 Comparison to related work 12
Background 15
3.1 Network Graphs 15
3.1.1 Connected Graph 15
3.1.2 Unconnected Graph 15
3.1.3 Directed Graph 15
3.1.4 Undirected Graph 15
3.2 Shortest Path Problem 16
3.2.1 Conventional shortest path algorithms 16
3.3 Machine Learning 18
3.3.1 Machine learning techniques 18
3.3.2 Supervised learning 18
3.3.3 Unsupervised learning 18
3.3.4 Semi-supervised learning 19
3.3.5 Reinforcement learning 19
3.4 Artificial Neural Network Algorithms 19
3.5 Graph neural network (GNN) 20
3.5.1 Graph network (GN) block 20
3.6 Graph embedding 23

3.6.1 Graph Embedding Types 24
3.7 Message Passing Neural Network (MPNN) 25
3.8 Routing and Wavelength Assignment (RWA) 25
3.8.1 Wavelength-Assignment Heuristics 26
GNN framework for Routing and Wavelength Assignment (RWA) 28
Graph Generation 29
Target graph attribute vector 29
Input graph attribute vector 30
ML output 31
Loss Computation and optimization 31
4.2 Network Routing Workflow (Test Phase) 32
Check mechanism 34
Path recovery 40
4.3 Wavelength Assignment 41
4.3.1 Traffic generation 42
4.3.2 Layered Graph Generation 42
4.3.3 Implementation of Heuristic Wavelength assignment methods 43
4.4 Performance evaluation metrics 46
4.4.1 Routing performance evaluation 46
4.4.2 Wavelength assignment performance evaluation 47
Results 49
5.1 ML Based Routing Results 49
5.1.1 Dataset Generation 49
5.1.2 Dataset Statistics 49
5.1.2 GNN Hyperparameters 50
Node/Edge classifier performance evaluation 52
Routing performance evaluation 58
5.2 Wavelength assignment numerical results 59
Conclusion 63
Future Work 64
Bibliography 65

List of Figures

Figure 1. Neuron model of ACNN [3]. 5
Figure 2. ACNN topology for SP problem. 5
Figure 3. An example of shortest path finding. 7
Figure 4. (a) Network topology; (b) state space representation and neural network architecture. 8
Figure 5. Schematic of the proposed 5-node network for RWA modeling. 10
Figure 6. Machine learning Routing Computation module workflow [7]. 12
Figure 7. Diagram of a multilayer network [9] 19
Figure 8. Updates in a GN block. 22
Figure 9. Each node embeds to d-dimensional embedding space [18] 24
Figure 10. Each node mapped to a low-dimensional feature vector [17] 24
Figure 11. A sample graph (Left). Mapping of the sample graph which shows the nodes that have close structure, are mapped close to each other in the embedding space (Right). 24
Figure 12: Model training blockdiagram 29
Figure 13. Graph generated by proposed algorithm in [21]. Geographical graph(right) with separated nodes, minimum spanning tree(middle), combined graph(left) 29
Figure 14: The original graph that the source and destination nodes are marked (Right graph), Routed path graph (middle graph), labeled graph showing the nodes and edges that are part of the path with “T” and others with “F” 30
Figure 15. A fully connected MLP with two layers and 32 neurons in each layers 32
Figure 16: Blockdiagram of proposed algorithm for routing 33
Figure 17. Graph network block 33
Figure 18. Decoder and Encoder block are using two independent MLPs for edges and nodes of the graph 33
Figure 19: Core block 34
Figure 20 Nodes 3 and 4 are labeled incorrectly, But the path has been obtained. 35
Figure 21: The shortest path is detected by both edge labels and node labels (Large nodes are solution labeled nodes, thicker edges are solution labeled edges). 38
Figure 22: The Shortest Path is detected by tracking node labels, while by following edge labels the shortest path will not be achieved (large nodes are solution labeled nodes, thicker edges are solution labeled edges) 38
Figure 23: The real shortest path(Left), the path detected by following the node/edge labels which is not the shortest one (Right) 39
Figure 24: The Path (not shortest) is detected by tracking node labels, while by following edge labels the path will not be achieved (large nodes are solution labeled nodes, thicker edges are solution labeled edges). 39

Figure 25: The path is not detected neither by following node labels nor edge labels. (large nodes are solution labeled nodes, thicker edges are solution labeled edges) 40
Figure 26: Path recovery 41
Figure 27: The real shortest path is in left graph, the output labels of our model is in middle image which shows the path has not detected using both node/edge labels. Path is recovered using path recovery algorithm (Right). 41
Figure 28: Random WA and Dijkstra routing block diagram 43
Figure 29: GNN routing, Layered graph, Random Wavelength assignment block diagram 44
Figure 30: First-fit WA and Dijkstra routing block diagram 44
Figure 31: GNN routing, Layered graph, First-fit Wavelength assignment block diagram 45
Figure 32: Least-used WA and Dijkstra routing block diagram 45
Figure 33: GNN routing, Layered graph, Least-used Wavelength assignment block diagram 46
Figure 34: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp1. 53
Figure 35: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp1. 53
Figure 36: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp2. 54
Figure 37: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp2. 54
Figure 38: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp3. 55
Figure 39: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp3. 55
Figure 40: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp4. 56
Figure 41: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp4. 56
Figure 42: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp5. 57
Figure 43: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp5. 57
Figure 44: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp6. 58
Figure 45: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp6. 58
Figure 46: PDF of arrival times of the lightpaths 60
Figure 47: PDF of holding times of the lightpaths 60
Figure 48: Blocking probability against number of requests 61

List of Tables

Table 1: Solved problems and ML models 13
Table 2: Update functions and their implementations 21
Table 3: Aggregation functions and their implementations 21
Table 4: Labeling of nodes in target graph 30
Table 5: Labeling in MPNN 31
Table 6: Change the MP steps 50
Table 7: Change the learning rate 51
Table 8: Change the number of neurons 51
Table 9: Fixed and Adaptive LR with 2 hidden layers 52
Table 10: Fixed and Adaptive LR with 3 hidden layers 52

Chapter 1

Introduction

1.1 Motivation

Applying machine learning (ML) to computational challenges is prevalent in numerous areas in computer science (AI, computer vision, graphics, NLP, comp-bio, and beyond). Computer networking, in contrast, has largely withstood the ML tide until recently. Recent advances suggest that this might be changing and recently, leveraging the methodologies of ML, several complex networking tasks can be performed with high accuracy and with limited or even without any human intervention.

Such a disruptive concept is expected to extend to any form of telecommunications network, irrespective of its topology, implementation or underlying technology, and may also benefit from network automation as allowed by the SDN principle [1]. Software-Defined Networking (SDN) is gradually becoming a reality and provides the ability to rethink and create highly programmable networks. SDN allows global-view networked datasets comprising routing, output, and configuration states to be collected and further optimized with ML / AI algorithms, providing a new range of opportunities to continuously enhance how network services are delivered and network resources allocated.

A detailed tutorial on Machine Learning algorithms, frameworks and applications to networking, including open challenges and concrete examples of Intelligent Networking can be found in [2].

SDN technology is ideally suited for implementing ML algorithms, or more precisely smart algorithms to speed up control behavior on the network. The key strength of this new paradigm is the possibility of applying various network optimization algorithms, each of which targets a different cost function. Furthermore, according to the centrality of the control plane, it is possible to train various ML algorithms simultaneously in off-line mode, and this can only be implemented after generalization and testing of the model. This process can be repeated if the model needs retraining, following the evolution of the network behavior itself. SDN, in short, allows smart control and configuration activities in a very short time and is a core component of future

telecommunications networks. In fact, 5G and the rise of new services (i.e., IoT, connected vehicles, Augmented and Virtual Reality AR / VR, etc.) are expected to make traffic matrices much more dynamic than they are today, thus requiring frequent network reconfiguration to better adapt the network resources to the actual traffic needs.

In this Thesis, we explore how to use ML in the context of optical WDM networks to target the Routing and Wavelength Assignment (RWA) problem. Performing RWA corresponds to assigning resources to each of the demands in a given traffic matrix, consisting of dedicated wavelength(s) along a physical pathway between two end-points. Usually RWA is solved in two key uses: 1) the architecture of an optical network to assess the amount of resources needed to be deployed under certain traffic forecasts; 2) The reconfiguration of the optical network, where the distribution of current network resources is re-optimized, caused by some complex changes in traffic, following some optimization goal usually aimed at preventing traffic congestion, underuse of resources, improved energy efficiency, etc.

RWA is formulated and solved in small or medium-sized network topologies using Integer Linear Programming (ILP), which provides optimal (e.g., cost-minimized) solutions to the detriment of complex, time-consuming, and intensive computations since the problem is known to be NP-hard. The literature has proposed suboptimal heuristic algorithms in large network topologies to speed up the RWA procedure.

In this Thesis, we transform the RWA problem into an ML-based classification problem, where the routing solution is provided by a classifier in response to a given input graph. To this end, a Message Passing Neural Network which is a type of Graph Neural Network (GNN) is trained based on various types of graphs. Once trained, such a classifier is able to provide a route for newly-incoming traffic requests in an online fashion, offering an RWA configuration within a few milliseconds, thus allowing to perform dynamic network adaptation and reconfiguration in response to frequently changing traffic patterns.

1.2 Thesis outline

The remainder of this work is organized as follows:

Chapter 2 provides a summary of the literature that by other researchers from Machine learning prospective and its application in optical networks for routing. It provides a general overview of the current research directions and highlights similarities and dissimilarities with respect to our work.

Chapter 3 reviews a summary of the key concepts to understand the theory on which the methods, approaches and models we use to perform experiments are based. We provide a basic overview about Graphs and explanation about basic Path finding algorithms. In the following, we

have a summary of ML and introduce its various methods. Then we study the topic of Graph neural networks (GNN) and Embedding methods in more detail. At the end of the chapter, we describe the message passing neural network (MPNN) used in our proposed method and review different routing and wavelength assignment approaches.

Chapter 4 describes the framework which is exploited in our experiments in detail. We discuss the assumptions on the creation of the graphs in the training and testing datasets.

Chapter 5 surveys the outcomes got from the numerical examination, first giving a portrayal of the datasets and afterward concentrating on the assessment of the metrics in the diverse considered scenarios.

Chapter 6 concludes this thesis with a summary of its main contributions. Furthermore, it contains other considerations based on the developed experiments.

Chapter 7 gives some prospective points for the future work of this research.

Chapter 2

Related works

This chapter describes recent studies focusing on the use of machine learning techniques in op tical network routing and wavelength assignment. As the thesis constitutes two parts shortest path finding problem and wavelength assignment. In the first part we provide the description of recent studies that focus on ML-based solutions for the shortest path problem and then we present the studies that are using Machine Learning techniques in routing and wavelength assignment. A short comparison of the mentioned techniques in relation to the work developed in this thesis will be underlined at the end of this chapter.

2.1 ML-based shortest path computation

The base of many graph algorithms and applications lies in finding the shortest path between nodes. Traditional specific approaches like breadth-first-search (BFS) do not scale up to contemporary, rapidly emerging vast networks of today. Therefore, approximation methods must be found to allow scalable graph processing with a significant speed-up.

Here we review some ML-based methods to compute shortest path between two nodes in a graph.

2.1.1 Auto Computed Neural Network (ACNN):

In [3], a neural network model called auto-wave competed neural network (ACNN) for the SP problem has been proposed.

Firstly, in this study, the proposed algorithm synchronously updates the threshold of all neurons, i.e., it is a parallel algorithm. Secondly, only the maximum M paths are allocated at each neuron's threshold, which significantly reduces the memory space and the computation needed. Thirdly, while the M-paths limited scheme may often discard the optimal paths due to the randomness of the constraints and the rigidity of the cost function, the proportional selection scheme will provide ample opportunity to revive or survive the search process with the optimal auto-waves (i.e., paths).

The ACNN neuron consists of three parts, *i.e.*, the minimum selector, the auto-wave generator and the threshold updater, see Figure 1.

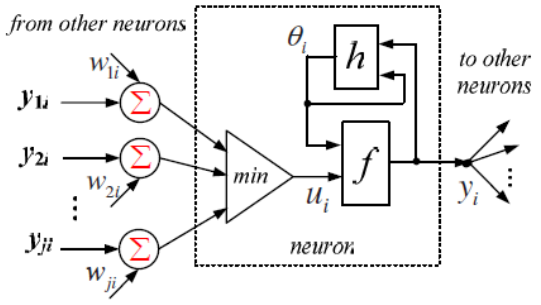


Figure 1. Neuron model of ACNN [3].

The ACNN neuron can be described with the following equations,

$$Z_i(t) = \{j \mid W_{ji} \neq \infty \ \& \ \ y_j(t-1) > 0\} \qquad \textit{Eq. 1}$$

$$u_i(t) = \begin{cases} 0 & Z_i(t) = \emptyset \\ \min\limits_{j \in Z_i(t)} (y_j(t-1) + w_{ji}) & \text{otherwise} \end{cases} \qquad \textit{Eq. 2}$$

$$y_i(t) = f[u_i(t), \theta_i(t-1)] = \begin{cases} u_i(t) & u_i(t) < \theta_i(t-1) \\ 0 & \text{otherwise} \end{cases} \qquad \textit{Eq. 3}$$

$$\theta_i(t) = h[y_i(t), \theta_i(t-1)] = \begin{cases} \theta_i(t-1) & y_i(t) = 0 \\ y_i(t) & \text{otherwise} \end{cases} \qquad \textit{Eq. 4}$$

Where i is the index of neuron, t is the time (or gives the iterations). $\theta_i(t), u_i(t)$ and $y_i(t)$ are the threshold, internal activity the output of neuron i at time t respectively. w_{ji} is the weight of the connection from neuron i to j. $Z_i(t)$ is the set of neurons that are fired at time t and is reachable to neuron i.

An ACNN isomorphic for the weighted graph G should be constructed when applied to the SP problem, i.e., every G-node corresponds to a single network neuron, then w_{ij} is related to the weight of the edge(i, j) in G, see Figure 2.

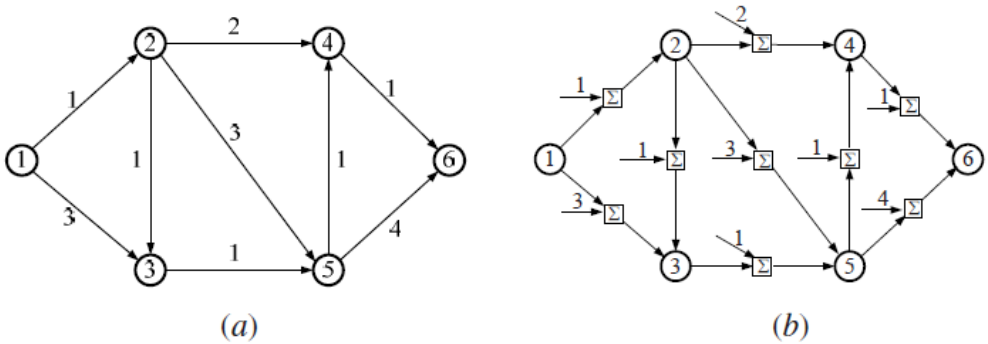


Figure 2. ACNN topology for SP problem. (a) A weighted digraph. The circles with numbers inner are the nodes, and the numbers on edges are the costs related to the corresponding edges. (b) The ACNN model for the SP problem of the graph is shown in (a). The circles with numbers inside are the neurons, and the squares with "∑" inside are the aggregators on the Corresponding links.

All neurons have infinite threshold and zero-internal-activity initializations. Fire the source neuron to run the network, and the firing would inspire a few propagating auto-waves across the entire network. When passing through the neuron i if it is the closest one, an auto-wave 's traveling distance would be recorded at the $\theta_i(t)$ threshold. All neurons gradually decrease their levels until the algorithm ends.

When the algorithm stops, the threshold $\theta_i(t)$ is equal to the distance from the source neuron to the neuron i of the shortest path.

The ACNN based shortest path algorithm is parallel, non-parameterized. For more detail about ACNN, the reader can refer to the reference [4].

2.1.2 Stochastic Shortest Path-based Q-learning (SSPQ)

In [5], a method based on reinforcement learning (RL) is used to find the shortest path for autonomous robots. RL has been recently used as a mechanism for autonomous robots to learn pairs of state-action by interacting with their environment. Most RL methods, however, usually suffer from slow convergence in practical applications when deriving optimal policies. A stochastic, shortest path-based Q-learning (SSPQL) approach is proposed to solve this problem, integrating a stochastic shortest path-finding approach with Q-learning, a well-known model-free RL method.

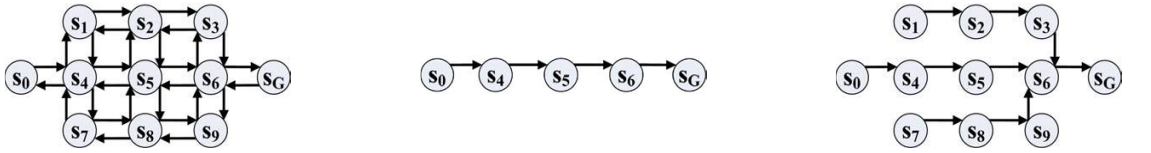
A learning method combining model-free and model-based RL methods is proposed in this paper to enhance both speed of learning and adaptability in dynamic environments. The Q-learning algorithm was adopted as the model-free RL method, and the model-based learning method used an addition of the SSP finding method.

Stochastic shortest path-finding method:

It may be useful for an external trainer to raise the probabilities of important state-action pairs to overcome the slow learning speed of model-free Qlearning. Although the trainer has only limited knowledge of the working environment, this can still be useful for improving speed of learning. A robot can learn incrementally from an internal state-transition model, and can infer from the model the essential state action pairs that make up the optimum local strategy. Then these essential state-action pairs will serve as the external trainer and be used to improve speed of learning. An SSP-finding method has been employed in the SSPQL to obtain these essential state-action pairs. The stochastic shortest path-finding method can then propose optimal local state-action pairs by using an internal state-transition model that has been incrementally learned by the robot. In the SSPQL these optimum local state-action pairs can be given greater selection probability by increasing the respective Q-value for each state-action pair. Given an experienced

state-transition model shown in Figure 3(a), the single-pair shortest path from the initial state, s_0 , to the goal state, s_G, is shown by Figure 3(b).

A * is a well-known successful heuristic search tool for this type of problem. Nevertheless, the purpose is to improve the probability of any action from the internal model that can be obtained. Here, Dijkstra algorithm [6] is used to find shortest stochastic paths based on the expected cost.



(a) An example of a state transition model. *(b) A shortest path from s_0 to s_G.* *(c) Shortest paths from all states to a goal state.*

Figure 3. An example of shortest path finding.

SSP-finding method with Q-learning:

At the same time, the SSPQL algorithm learns and uses the state transformation model by integrating the SSP method with the Q-learning method.

A prior observed state, a previous action, and a present state must be stored in a probabilistic state-transition model at each stage to learn the state-transition model. The state-transition model can be nearly empty in the early stages of learning. As a robot uses an exploration strategy primarily influenced by the Q-value and the exploration bonus to explore its environment, it will begin learning the state-transition model. Using the SSP method with an integrated cost function as shown in (2) in the stochastic state-transition model, the shortest paths that reach this goal state from all states can be found after the robot reaches a target state.

In summary, action-selection in SSPQL is based on a linear combination of Q-value, SSP-value, and the value of exploration bonus, which is given by

$$a = argmax\left[w_Q Q(s', a') + w_{SSP}\, Q_{SSP}(s', a') + w_{EXP}\, Q_{ExP}(s', a') + \varepsilon\right] \qquad \textit{Eq. 5}$$

Here, $w_Q$is known as weight of Q-learning, w_{SSP} is a weight value of SSP , $w_{EXP}$is a weight of the exploration bonus, and ε is a random number for exploration. $Q(s', a')$, $Q_{SSP}(s', a')$ and $Q_{ExP}(s', a')$ have different training quality characteristics.

Convergence to optimality can be ensured by using $Q(s', a')$. However, when using only the Q-value the learning speed is very slow. On the other hand, SSP $Q_{SSP}(s', a')$ will affect a speed of rapid convergence. The method of discovering SSP is a batch process while Q-learning is an incremental process. Hence, $Q_{SSP}(s', a')$ may have significant value in the early stages of learning to achieve the target level.

Lastly, $Q_{ExP}(s', a')$ will take an agent to unexplored conditions. If an agent enters a state, Q_{ExP} may experience a rapid decline. So in the first episode, Q_{ExP} mainly affects the learning efficiency.

2.2 ML techniques in routing and wavelength assignment (RWA)

Programmable optical networks make it possible to create general transport infrastructure capable of accommodating a wide range of 5G services [7]. In such a scenario, one of the main goals of an Infrastructure Provider (InP) is to maximize profit by making the most efficient use of infrastructure resources, e.g. by operating with link utilization values close to the capacity crunch. Furthermore, using the same transport network between Connectivity Services (CSs) with different Quality of Service (QoS) constraints (i.e., priorities) and different revenue values is a daunting challenge, i.e. the routing strategy should be able to include as many CSs as possible (to optimize revenue) while maintaining the necessary QoS restrictions.

2.2.1 ML-Based Quality of Service Routing

The study in [8] proposes a routing policy based on Reinforcement Learning (RL), aimed at maximizing an infrastructure provider's profit. This aim is accomplished by carefully selecting (i) whether to accept a new CS request or not, and (ii) which path to assign to those accepted CSs. All this is done while taking into account each CS request's QoS constraints and revenue level. Results show that the proposed strategy will raise the profit margin by up to 13 % relative to conventional heuristics when the network operates under high resource usage conditions.

This is called a programmable optical network system where a network manager has full knowledge of where and how many wavelength resources are being used. The controller is also responsible for taking decisions on the provisioning of CS with the goal of optimizing the InP benefit. Two types of CSs are considered: (i) HP, i.e. with strict latency specifications that can only be supplied over the shortest distance between the source and the destination node, and (ii) LP, i.e. best-effort CSs that can be supplied with appropriate communication resources over any path. HP CSs generate higher profits than LP ones.

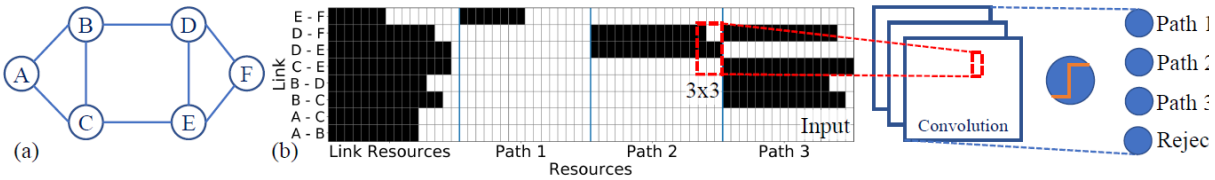


Figure 4. (a) Network topology; (b) state space representation and neural network architecture.

The study proposes an RL-based routing strategy modeled on a Policy Network (PN), which uses a neural network to reflect a stochastic process. For a certain network state associated with a particular CS request, the PN discovers that taking an action would result in a reward, which the reward should be maximized. Considering the topology presented in Figure 4(a), Figure 4(b) shows how the state space is organized (e.g., for a hypothetic CS request from node E to node F), in addition to the NN architecture considered in this study. Assuming link capacity of 16 wavelengths on each link, the first block of Figure 4(b) indicates how wavelength resources are used on each link, where "1" (i.e., black) corresponds to the used wavelength and "0" (i.e., white) corresponds to the free wavelength. The following k=3 blocks in Figure 4(b) represent the state of the links of each of the candidate paths from node E to node F should they be chosen for provisioning the CS. If upon a CS request a route does not have enough wavelength capital, it is excluded from the state space. State space is used as an input to the neural network that has a convolutional layer with a 3-by-3 2D convolution window with the intention of extracting the best features from state space.

Every output is related to a particular action, i.e., "select" one of the available paths or "reject" the CS request.

To sum up, the paper introduced an RL-based routing strategy, modeled as a PN that can help increase an InP's profit margin. Results indicate that it is possible to increase the profit margin by up to 13% relative to traditional routing policies by proactively opposing low revenue CS in the scenario considered.

Supervised Machine Learning-Based Routing and Wavelength Assignment

In [1], a machine learning approach used to develop optical WDM networks has been introduced the framework included three phases: data generation, simulation and implementation of SDNs.

The authors have introduced NetGen, a scalable tool for building networking labeled data sets, for the data generation process. This tool wraps the Net2Plan tool 's regular functioning to scale and speed up its behavior.

With respect to machine learning, they have turned the well-known problem of routing and wavelength assignment into a Supervised Learning problem that can be solved using traditional ML algorithms. In particular, a logistic regression and Deep Neural Networks (DNNs) are trained with a ground-truth dataset of thousands of traffic matrices and their related RWA solutions given by the RWA ILP or First-Fit heuristic (the labels in our classification problem).

In general, DNNs provide useful non-linear learning structures applicable in the context of an optical WDM network for learning RWA structures associated with traffic matrices. For these

DNNs to learn to apply RWA configurations effectively, they need to be fed with enough data examples and a reduced number of RWA classes to avoid the so-called dimensionality curse.

Figure 5. Schematic of the proposed 5-node network for RWA modeling. The network models offline training phase and online prediction phase [7].

Turning RWA into a Supervised Classification Problem

Consider 5-node network topology in the Figure 5 and let's presume that each connection is fitted with a number of optical transponders allowing each connection to use W wavelengths per link (in what follows, W lambdas @ C Gbps).

Consider a 5×4 Traffic Matrix (TM) where elements d_{ij} (in Gbps) denote the traffic request from source i to destination node$j$ ($i \neq j$). The RWA algorithm generates a list mapping each traffic request d_{ij} (input) to a sequence of links and wavelength assignments (output). In the network of Figure 5 .

For example, demand d_{12} from source node 1 to destination node 2 uses the direct edge e_{12} and the first wavelength λ_1. Similarly, demand d_{54} continues the route defined by edges $e_{52} - e_{23} - e_{34}$ and uses the third wavelength λ_3. This configuration of routing and wavelengths (RWC) only applies to that particular matrix of demand. In other words, the RWA receives a serialized traffic matrix with all requests for traffic from source-destination as input and outputs its optimal RWC list:

- Input: $TM_1 = \{d_{12}, d_{13}, \ldots, d_{54}\}$
- Output: $RWC_1 = \{(e_{12}, \lambda_1), \ldots, (e_{52}, e_{23}, e_{34}, \lambda_3)\}$

For that particular Traffic Matrix TM_1, the output mark $y_1 = RWC_1$ is obtained by solving the RWA ILP. If the ILP is run again for a second TM_2 traffic matrix, then a new optimal RWC should

be obtained, namely $y_2 = RWC_2$, but maybe the previous RWC_1 is still true to satisfy all traffic demands while meeting the RWA constraints. In general, if 10,000 different traffic matrices are fed to the ILP, the ILP might theoretically generate up to 10,000 different optimal RWCs. In realistic situations, however, several RWCs refer to multiple input traffic matrices, thus reducing RWC space under consideration.

Consequentially, the RWA problem can be converted into a multi-class classification problem in which the input variables are the elements of an episodic traffic matrix and the output labels are the complete network routing and wavelength configuration (RWC) collection as obtained from the ILP solution. A classical supervised ML algorithm can then be trained using this dataset. Essentially, when fed with enough data examples (both traffic matrices and the corresponding optimal RWCs), the ML algorithm should be able to generalize and generate an optimal RWC on a new unseen traffic matrix, thus accurately performing RWA without the need to solve the ILP. In addition, the ML model should learn from the input data provided, i.e. if the ML model is fed with data manually configured from an operator or from some other algorithm (i.e., heuristics) instead of supplying RWC solutions from an ILP, the ML algorithm should also be able to replicate this way of solving RWA, thereby mimicking the algorithm it learned from.

As far as the number of RWC classes is concerned, getting too many of them will prevent the ML algorithm from learning the data patterns correctly due to the so-called dimensionality curse. Therefore, in order to reduce the number of RWCs in use, we carried out a forward evaluation of RWC to reduce the number of classes by assigning new TMs with more frequent RWCs as long as they provide a feasible solution and satisfy some minimum requirements in terms of average network load and hop count.

The authors have developed the Netgen tool, built on Net2Plan1 planner tool, for database generation. Two of the most common ML algorithms have been trained and tested once the datasets have been developed, namely: Logistic Regression (LR) and Deep Neural Network (DNN) feed-forwards.

Logistic Regression [9] is a simple linear classifier that, together with a softening sigmoid function, adjusts a linear regression to the data. Because of their linear nature, logistic regression classifiers are very fast to train and interpret but prone to underfitting. In comparison, Deep Neural Networks [9] showed great success in a wide range of scenarios due to the fact that they stack linear neuron layers coupled with non-linear activations, thereby generating non-linear classifiers. The Machine learning Routing Computation workflow module can be seen in the figure below.

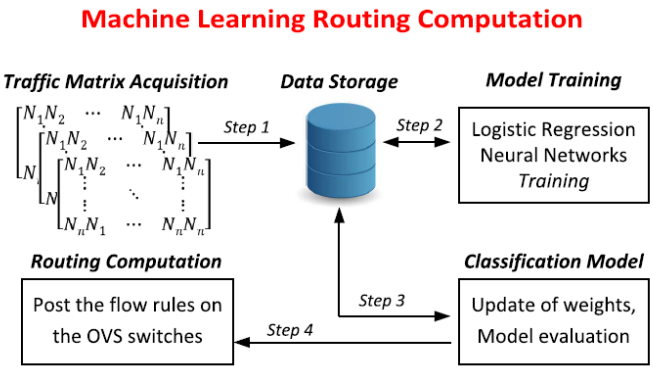


Figure 6. Machine learning Routing Computation module workflow [7].

The authors have finally shown how SDN and ML can come together through the Machine Learning Routing Computation Module (MLRC) to push path provisioning through an SDN network. Using this module, the Routing and Wavelength Configurations obtained via ML can be delivered quickly into an SDN network. As a result, the combination of Machine Learning and SDN paradigms to facilitate traffic-aware and responsive networks, the ability to react very quickly to changes and implement such changes in configuration easily through SDN has advanced. Their findings demonstrate how their approach reduces the upgrade time needed for the network configuration.

2.3 Comparison to related work

In this thesis, we divide the RWA problem into routing and wavelength assignment problem. ML-based routing has been used in our work which showed good performance in large optical networks where heuristic algorithms have not been so fast, then we used some conventional algorithm in order to assign wavelength to each traffic request.

In the routing part we have exploited a Node and Edge Embedding model and our model includes three components: An "Encoder" graph net, which independently encodes the edge, and node attributes. A "Core" graph net, which performs N rounds of processing (message-passing) steps. A "Decoder" graph net, which independently decodes the edge, node attributes (does not compute relations etc.), on each message-passing step.

The model is trained by supervised learning. Input graphs are procedurally generated, and output graphs have the same structure with the nodes and edges of the shortest path labeled (using 2-element 1-hot vectors). Therefore at the output of the model we will have labeled nodes and edges showing whether they are part of shortest path or not. Then we check the correctness of the path detected by the model and execute an extra processing for undetected paths in order to have a valid path between node pairs.

For the wavelength assignment sub-problem, a number of heuristics have been proposed such as Random Wavelength Assignment (Random Fit), first-fit, least used, most used, least loaded. The aim of this step of processing is allocating wavelength to each traffic request to minimize blocking probability and maximize the network utilization.

In [4] [5] the authors have used machine learning in order to find a path between a pair of nodes and they can work well for the small networks, But their algorithms is not well suited for large networks.

In [8] the authors have used RL-based RWA for different connectivity services with different priority levels. Here we try to balance the traffic load over the whole network using conventional algorithms.

In the Table 1 we have summarized the different scenarios illustrated in the previous subsections and underlined their methods in order to solve the shortest path (SP) problem or routing and wavelength assignment problem (RWA).

Table 1: Solved problems and ML models

Authors	problem	ML model
Jiyang Dong et al. [4]	SP	• Artificial Neural Network
Woo Young Kwon et al. [5]	SP	• Reinforcement learning (RL)
Carlosns Natalino et al. [8]	RWA	• Convolutional Neural Network (CNN) • Reinforcement Learning (RL)
Ignacio Martín et al. [1]	RWA	• Deep Neural Network • Logistic Regression
Our work	RWA	• Graph Neural Network • Heuristic Algorithms

Chapter 3

Background

In this chapter we provide detailed information about the algorithms and methods which we will be adopted in our framework.

3.1 Network Graphs

Let $G = (V, E)$ denote a graph with node feature vectors X_v for $v \in V$ and with edge feature vectors X_e for $e \in E$.
There may be different types of Graphs:

- Connected Graph
- Unconnected Graph
- Directed Graph
- Undirected Graph

3.1.1 Connected Graph

If there exists at least one branch between any of the two nodes of a graph, then it is called as a connected graph. That means, each node in the connected graph will be having one or more branches that are connected to it. So, no node will be isolated or separated.

3.1.2 Unconnected Graph

When there is at least one node in the network where even a single branch remains unconnected, then it is called as an unconnected network. Therefore, an unconnected graph may contain one or more isolated nodes.

3.1.3 Directed Graph

If all the branches of a graph are represented with arrows, then that graph is called as a directed graph. These arrows indicate the direction of current flow in each branch. Hence, this graph is also called as oriented graph.

3.1.4 Undirected Graph

When a graph's branches are not represented with arrows, the graph would then be called as an undirected graph. Because current flow directions do not exist, this graph is often named as a non-oriented graph.

3.2 Shortest Path Problem

The shortest path problem consists in finding the shortest path or route from a source point to a final destination. In general, we use graphs in order to represent the shortest path problem. A graph is an abstract mathematical entity, containing collection of nodes and edges. Edges connect pairs of nodes. Along the edges of a graph it is possible to move from one node to other nodes. Based on whether or not one can walk either sides around the edges, or only one side decides whether the graph is a directed graph or an undirected graph. In addition, edge lengths are sometimes referred to as weights, and the weights are typically used to measure the shortest path from one point to another. [10]

3.2.1 Conventional shortest path algorithms

Dijkstra Algorithm

For each node within a graph we assign a label that determines the minimal length from the source point s to other nodes v of the graph. In a computer we can do it by defining an array $d[]$.The algorithm works sequentially, and tries to decrease the node label value in each step. The algorithm stops when all nodes have visited. The label at the source point s is equal to $(d[s] = 0)$; hence, labels in other nodes v are equal to infinity $(d[v] = \infty)$, which means that the length from the starting point s to other nodes is unknown. We may use a very large number on a machine simply to represent infinity. Furthermore, we will define for each node v whether or not it has been visited. For this reason, we declare an array of Boolean type called $u[v]$, where initially, all nodes are assigned as unvisited (u[v] = false). [11]

The algorithm of The Dijkstra consists of n iterations. Once all nodes are visited, the algorithm ends; otherwise, from the list of unvisited nodes, we have to pick the node with the lowest (smallest) value on its label (we must choose a starting point s at the beginning). Before that, we will consider all the neighbors of this node (Neighbors of a node are those nodes with the initial node having similar edges). We will consider a new length for each unvisited neighbor equal to the sum of the value of the mark at the initial node v (d[v]) and the length of edge l connecting them. If the resulting value is less than the label value, then with the newly obtained value [3] we have to change the value in that label.

$$d[neighbors\,] = min\,(\,d\,[\,neighbors\,]\,,d[\,v\,] + l\,) \qquad \textit{Eq. 6}$$

After repeating this step n times, all graph nodes are visited, and the algorithm begins or continues. The nodes that aren't linked to the starting point will remain assigned to infinity. To restore the shortest path from the starting point to other nodes, we need to identify array p[] where the number of nodes p[v], which penultimate nodes in the shortest path, will be stored for each node. In other words, a complete path from s to v is equal to the statement below [5].

$$P = (s, \dots, p[p[p[v]]], p[p[v]], p[v], v) \quad \textit{Eq. 7}$$

Floyd-Warshall Algorithm

Consider the graph G, where nodes were enumerated from 1 to n. d_{ijk} means the shortest path from i to j, which also passes through node k. Obviously if there is a edge between nodes i and j it will be equal to d_{ij0}, otherwise it will be set to infinity. However, there can be two choices for other values of d_{ijk}: (1) If the shortest path from i to j doesn't pass from the node k then value of d_{ijk}will be equal to d_{ijk1}. (2) If the shortest path from i to j passes from the node k then first it goes from i to k, after that goes from k to j. In this case the value of d_{ijk} equals to $d_{ikk-1} + d_{kjk-1}$. And to determine the shortest path we just need to find the minimum of these two statements [11]:

$$d_{ij0} = \textit{the length of edge between nodes } i \textit{ and } j \quad \textit{Eq. 8}$$

$$d_{ijk} = \min(d_{ijk} - 1, d_{ikk} - 1 + d_{kjk} - 1) \quad \textit{Eq. 9}$$

Bellman-Ford Algorithm

The Bellman-Ford algorithm admits edges with negative weights as compared with the Dijkstra algorithm. A graph may therefore contain loops of negative weights which will produce multiple paths from the starting point to the final destination, where each cycle will minimize the length of the shortest path.

3.3 Machine Learning

This section summarizes some of the most popular algorithms commonly classified as machine learning. The literature on ML is so vast that the possibilities of this section go far beyond just a brief description of all the major ML approaches. In this section, however, we provide a detailed view of the main ML techniques that are used in the work we refer to in the rest of this thesis. Here we provide some basic insights for the reader that might help better understand the remaining parts of this thesis.

3.3.1 Machine learning techniques

We categorize the algorithms in three main classes, described in the next sections: supervised learning, unsupervised learning and reinforcement learning. Semi-supervised learning, and unsupervised learning, are also introduced. ML algorithms have been applied successfully to a large range of problems. Before delving into the different ML methods, it is worth pointing out that more than a decade of research on the application of ML techniques to wireless networks has taken place in the context of telecommunication networks.

3.3.2 Supervised learning

Supervised learning is used in a variety of applications including speech recognition, spam identification, and object recognition. The goal is to predict the value of one or more output variables given the value of an input variables vector x. The output variable may be either a continuous (regression problem) or a discrete (classification problem). A training data set contains N samples of the input variables and the related output values. Different methods of learning build a function y(x) which allows to predict the value of the output variables in corresponding to a new input value [12].

Supervised learning is simply a formalization of the learning process through examples. The learner (typically a computer program) is learning in supervised learning, supplied with two data sets, a training set and a test set. The idea is to "learn" from a set of labeled examples in the training set to identify unlabeled examples in the test set with the highest possible accuracy [9].

3.3.3 Unsupervised learning

Unsupervised learning is based on the method that can be viewed as a teacher's absence, and thus absolute error steps. It's useful when learn how to cluster or group elements is needed. Elements can be grouped (clustered) according to their similarity. Data is unlabeled, not classified in unsupervised learning and the algorithms of the program operate upon the data without prior training. Unsupervised learning algorithms can perform tasks that are more complex than supervised learning algorithms. They includes clustering that can be done by K means clustering,

hierarchical, and hidden Markov model. Network analysis, and market research are among the most successful applications of unsupervised learning methods.

3.3.4 Semi-supervised learning

Semi-supervised learning blends labeled and unlabeled examples to construct a function or classifier relevant to it.

3.3.5 Reinforcement learning

The RL model encourages agents to learn using only an evaluative input, referred to as the incentive, to explore the available behaviors and improve their behavior. The agent's goal is to optimize its output over the long term. The agent therefore not only takes the immediate benefit into account, but considers the impact of his actions on the future. The two most critical features of RL are delayed reward and trial-and - error [12].

3.4 Artificial Neural Network Algorithms

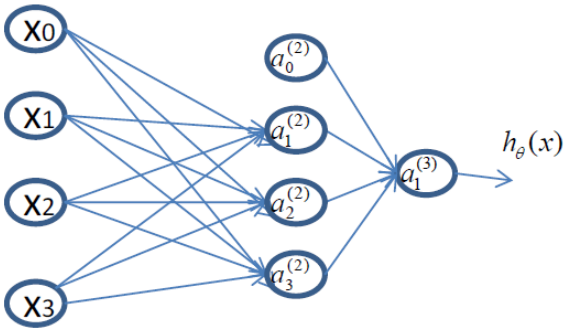


Figure 7. Diagram of a multilayer network [9]

Like the human brain, artificial neural networks are built, with neuron nodes interconnected like web. The human brain comprises hundreds of billions of neurons called cells. Each neuron consists of a cell body responsible for information processing by transferring information to (inputs) and (outputs) from the brain [9].

An ANN may have hundreds or thousands of artificial neurons, called processing units, linked via nodes. These processing units are composed of units for input and output. The input units obtain different types and knowledge structures based on an internal weighting scheme, and the neural network aims to learn about the information provided in order to generate one output result. Just as humans need rules and guidelines to produce a result or output, ANNs also use a set of learning rules called back-propagation, an abbreviation for backward error propagation, to improve the results of their output.

An ANN initially goes through a phase of training where it learns to recognize patterns in the data, whether visually, aurally or textually. During this supervised phase, the network compares its actual output to what it was supposed to produce the desired output. Through back-propagation the difference between the two tests is balanced. This means the network works backwards, going from the output unit to the input units to adjust the weight of its connections between the units until the difference between the actual and desired result produces the lowest error possible.

3.5 Graph neural network (GNN)

GNNs have originally been proposed for learning on graph structures [13]. Their structures consists of a message passing scheme [14], where the representation $h_s^{(k)}$ of each node s (in iteration k) is recursively updated by aggregating the representation of neighbor nodes. GNNs can be adopted for reasoning by considering objects as nodes and assuming all objects pairs are connected, i.e., a complete graph [15]:

$$h_s^{(k)} = \sum_{t \in S} MLP_1^{(k)}(h_s^{k-1}, h_t^{k-1}), h_s = MLP_2\left(\sum_{s \in S} h_s^{(k)}\right), \quad \textit{Eq. 10}$$

Where h_s is the answer/output and K is the number of GNN layers. Each object's representation is initialized as $h_{(0)} = X_s$. Although other aggregation functions are proposed, we use sum in our experiments. Each node aggregates feature vectors of all its neighbors to compute its new feature vector. After k number of iterations of aggregation, a node will be represented by its transformed feature vector, which shows the structural information within the node's k-hop neighborhood. The representation of a complete graph can then be obtained through pooling, for instance, by summing the representation vectors of all nodes within the graph.

In the rest of the sub-segments, we present our graph networks framework, which sums up and expands a few professions around there.

3.5.1 Graph network (GN) block

A motivating example is introduced to help make more concrete formalism to the GN. Imagine a collection of rubber balls in an arbitrary gravitational field predicting the motions, which rather than bouncing against each other, each has one or more springs linking them to some (or all) of the others.

One graph is represented as a 3-tuple G = (u; V; E) within our GN system. The u is a global attribute; here, u could represent the gravity field. The $V = \{v_i\}(i = 1: N^v)$ is the set of nodes (of cardinalityN^v), where the element of each v i is a node. V might represent each ball, for example, with attributes for position, velocity, and mass. For instance, V can represent each ball, with attributes for position, velocity, and mass. The E=$\{(e_k, r_k, s_k)\}_{k=1:N^e}$ is the set of edges (of

cardinalityN^e), and each e_k is the edge's attribute, r_k is the index of the receiver node, and s_k is the index of the sender node. For example, E might represent the presence of springs between different balls, and their corresponding spring constants. [16]

Internal structure of a GN block

A GN block is based on three update" functions, ∅ which are implemented by Multi-Layer Perceptron (MLP) (**Table 2**).

Table 2: Update functions and their implementations

Update functions	Implementation
$\mathbf{e'_k} = \emptyset^{e}\left(\boldsymbol{e_k}, \boldsymbol{v_{r_k}}, \boldsymbol{v_{s_k}}, \boldsymbol{u}\right)$	$MLP_e([e_k, v_{r_k}, v_{s_k}, u])$
$\boldsymbol{v_i'} = \emptyset^{v}\left(\bar{\boldsymbol{e}}_{\boldsymbol{i}}', \boldsymbol{v_i}, \boldsymbol{u}\right)$	$MLP_v([\bar{e}_i', v_i, u])$
$\boldsymbol{u'} = \emptyset^{u}(\bar{\boldsymbol{e}}', \bar{\boldsymbol{v}}', \boldsymbol{u})$	$MLP_u(\bar{e}', \bar{v}', u)$

And three aggregation" functions, ρ which are implemented by Sum function (**Table 3**).

Table 3: Aggregation functions and their implementations

Aggregation functions	Implementation
$\bar{\boldsymbol{e}}_{\boldsymbol{k}} = \boldsymbol{\rho}^{\boldsymbol{e}\rightarrow\boldsymbol{v}}(\boldsymbol{E_i'})$	$\sum_{\{k:r_k=i\}} e_k'$
$\bar{\boldsymbol{e}}' = \boldsymbol{\rho}^{\boldsymbol{e}\rightarrow\boldsymbol{u}}(\boldsymbol{E}')$	$\sum_i v_i'$
$\bar{\boldsymbol{v}}' = \boldsymbol{\rho}^{\boldsymbol{v}\rightarrow\boldsymbol{u}}(\boldsymbol{V}')$	$\sum_k \bar{e}_k'$

Where $E_i' = \left\{(\mathrm{e'}_k, r_k, s_k)\right\}_{k=1:N^e}$, $V' = \{v_i'\}_{i=1:N^v}$ and $E' = U_i\, E_i' = \left\{(\mathrm{e'}_k, r_k, s_k)\right\}_{k=1:N^e}$.

The $\emptyset^e$ is applied for all the edges in the graph to update, and the $\emptyset^v$ function is mapped over all nodes in the graph to compute per-node updates, but the $\emptyset^u$ is just applied once for the global update. The ρ functions take a set of inputs, and reduce it to a single element that represents the aggregated information. The ρ functions should be invariant to permutations of their inputs, and should take variable numbers of arguments as input (e.g., element wise summation, mean, maximum, etc.).

Computational steps within a GN block

When a graph, G, is given as input to a GN block, the computations starts from the edge, then the node, and finally the global level. Algorithm 1 shows the following steps of computation:

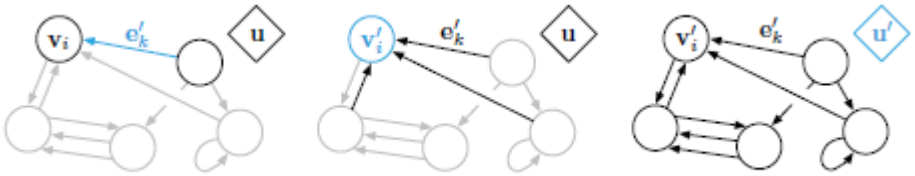
Algorithm 1. *Computational steps within a GN block*

function GRAPHNETWORK(E, V, $\mathbf{u}$)
  for $k \in \{1 \dots N^e\}$ do
    $\mathbf{e}'_k \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$ ▷ 1. Compute updated edge attributes
  end for
  for $i \in \{1 \dots N^n\}$ do
    let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\, k=1:N^e}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\to v}(E'_i)$ ▷ 2. Aggregate edge attributes per node
    $\mathbf{v}'_i \leftarrow \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$ ▷ 3. Compute updated node attributes
  end for
  let $V' = \{\mathbf{v}'\}_{i=1:N^v}$
  let $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$
  $\bar{\mathbf{e}}' \leftarrow \rho^{e\to u}(E')$ ▷ 4. Aggregate edge attributes globally
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\to u}(V')$ ▷ 5. Aggregate node attributes globally
  $\mathbf{u}' \leftarrow \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u})$ ▷ 6. Compute updated global attribute
  return $(E', V', \mathbf{u}')$
end function

Edge Update and Aggregation:

Step 1: $\emptyset^e$ Is applied over all the edges in the graph, and its arguments are$(\mathrm{e}_k, v_{r_k}, v_{s_k}, \mathrm{u})$. It returns e'_k in our work, this might correspond to the information about each node (position and etc.). The set of resulting per-edge outputs for each $node_i$, is $E_i{}' = \{(\mathrm{e}'_k, r_k, s_k)\}_{r_k,=i\,,k=1:N^e}$. And $E' = U_i\, E'_i = \{(\mathrm{e}'_k, r_k, s_k)\}_{k=1:N^e}$ is the set of outputs for all edges.

Step 2: $p^{e\to v}$ applies over $E'_i\ set$. It aggregates the edge updates for edges that project to $node_i$, into $\bar{e}'_i$. It will be used in the next step's node update.



(a) Edge update (b) Node update (c) Global update

Figure 8. Updates in a GN block. Blue indicates the element that is being updated, and black indicates other elements which are involved in the update [17].

Node Update and Aggregation:

Step1: $\emptyset^v$ Is applied over all the $node_i$ in the graph, in order to compute an updated node attribute. V' is the set of resulting outputs for each $node_i$, $V' = \{v_i'\}_{i=1:N^v}$.

Step2: $p^{e\to u}$ applies over E', and aggregates all of the edge updates, into $\overline{e}'$. Then it used in the next steps in order to global update.

Step3: $p^{v\to u}$ applies over V'. It aggregates all of the node updates, into $\bar{v}'$, then it used in the next step in order to global update.

Global Update:

$\emptyset^u$ Computes an update for the global attribute (u'). It is applied one time for the graph.

3.6 Graph embedding

Across different real-world applications, graphs such as social networks, word co-occurrence networks, and contact networks arise naturally. Analyzing those gives insight into the nature of culture, vocabulary and different communication patterns. Many approaches for carrying out the analysis have been proposed. Methods which use graph node representation in vector space have recently gained traction from the research community [18].
Graph analytical tasks can be abstracted broadly into the following four categories:

(a) *Node or link classification*: helps to evaluate the classification of nodes depending on other classified nodes and the network topology.
(b) *Link prediction*: Connection prediction refers to the role of forecasting missed connections or ties that might exist in the future
(c) *Clustering*: Is used to search, group and group subsets of related nodes
(d) *Visualization*: Gives insights into network structure.

Within the literature the term graph embedding was used in two ways:

1. To describe a whole graph in vector space.
2. To describe each particular node in vector space. We use this concept in our study as these representations can be used for tasks such as nodes and edges classification, differently from the previous interpretation (Figure 9) [18].

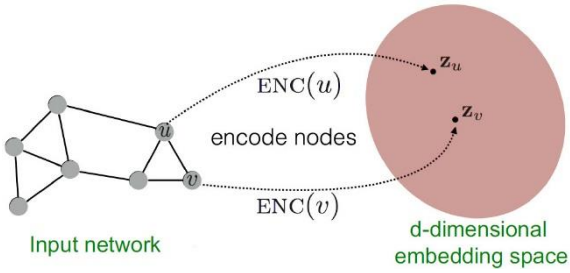


Figure 9. Each node embeds to d-dimensional embedding space [18]

An embedding therefore maps each node to a low-dimensional feature vector and tries to preserve the connection strengths between vertices (Figure 10). [17]

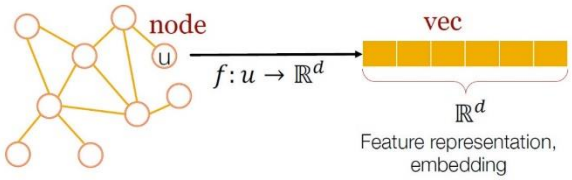


Figure 10. Each node mapped to a low-dimensional feature vector [17]

3.6.1 Graph Embedding Types

Position-aware node embedding: If the embedding of two nodes can be used to (approximately) recover their shortest path distance in the network, we call the node embeddings *"position-aware"*. This property is crucial for many prediction tasks.

Structure-aware node embedding: A node embedding is structure-aware if it is a function of up to q-hop network neighborhood of node v_i.

Graph neural Networks (GNNs) *are Structure-aware node embedding.* [17]

Figure 11. A sample graph (Left). Mapping of the sample graph which shows the nodes that have close structure, are mapped close to each other in the embedding space (Right).

As we can see in the above image, nodes v_1 and node v_2 have the same structure and their connections are close to each other, therefore by using Structural node embedding methods, they will map close to each other in the embedding space, but in fact their positions are not close to

each other and we cannot use this kind of node embedding methods to find shortest path, which is our main goal to route nodes in a network.

3.7 Message Passing Neural Network (MPNN)

Message passing Graph Neural Networks are a well-known type of GNNs that apply an iterative message-passing algorithm to propagate information between the nodes of the graph. MPNNs generalizes the concepts of Graph Neural Networks with two key insights.

Next, when computing the node embedding, instead of just aggregating computed messages from the local network neighborhood of a node, we require MPNNs to aggregate messages from others, which are neighbor nodes.

Second, when conducting message aggregation, the aggregation is clustered across all nodes instead of allowing each node to accumulate information individually, in order to differentiate nodes with different locations in the network. We construct MPNNs in such a way that each node embedding parameter corresponds to computed messages with respect to other nodes, thereby rendering the computed node embed position-aware.

3.8 Routing and Wavelength Assignment (RWA)

The problem of setting up lightpaths by routing and assigning a wavelength to each connection is called the Routing and Wavelength Assignment (RWA) problem.

Wavelength-Division Multiplexing (WDM) has increasingly gained popularity in optical fiber networks as a way of meeting the ever-increasing bandwidth demands of network users. End-users communicate with each other in a wavelength-routed WDM network via all-optical WDM channels, which are called lightpaths. In a wavelength-routed WDM network a lightpath is used to carry a traffic request, and it can span several fiber links. In the absence of wavelength converters, a light path will use the same wavelength over all the fiber connections it traverses; this property is known as the restriction of wavelength-continuity [1].

The problem concerns a network G = (V, E) where V is the set of nodes representing the physical network switches and E is the set of edges representing the physical network's fiber connections. Because of a set of requests for all-optical connections or lightpaths between node pairs and a collection of available wavelengths, the challenge is to find routes to their respective destination nodes from the source nodes and allocate wavelengths to those routes.

In WDM networks, the RWA issue can be classified into two forms based on traffic arrivals [19].

1. Static Lightpath Establishment (SLE): Traffic is static and the set of connection requests is identified beforehand. This sort of issue relates to the WDM network's planning

process. The proposed algorithms for solving the static RWA problem are called Offline algorithms.

2. Dynamic Lightpath Establishment (DLE) in the case of dynamic traffic: The traffic patterns change, and requests for connections arrive sequentially at random times over an infinite time period, one by one. A lightpath is set up when the traffic request arrives and released after a limited period of time. The DLE problem occurs in the operational process, in which each network resource needs to be handled effectively. The proposed algorithms to solve the dynamic RWA problem are called online algorithms.

3.8.1 Wavelength-Assignment Heuristics

The following heuristics for wavelength assignment have been proposed in the literature:

Random, First-Fit, Least-Used/SPREAD, Most-Used/PACK, Min-Product, Least Loaded. These heuristics can all be implemented using online algorithms and can be combined with different routing schemes as well. We describe the wavelength-assignment heuristics below [7].

Random Wavelength Assignment (R)

This scheme first scans the space of the wavelengths to identify the range of all usable wavelengths on the appropriate path. A wavelength is chosen randomly (usually with uniform probability) from the available wavelengths and assigned to the traffic request to be served.

First-Fit (FF)

All wavelengths are enumerated in this scheme. A lower-numbered wavelength is considered before a higher numerated wavelength when looking for available wavelengths. No global information is required in this scheme. Compared with Random Wavelength Assignment, this scheme's calculation cost is lower because there is no need to search the entire wavelength space for each route. The concept behind this scheme is to stack all of the in-use wavelengths into the lower end of the wavelength space, so that continuous longer paths to the higher end of the wavelength space are more likely to be available. This scheme works well in terms of blocking probability and fairness and is favored in practice because of its limited overhead computation and low complexity. As with Random, FF does not implement overhead communication since it needs no global information.

Least-Used (LU)/SPREAD

Least-Used chooses the least used wavelength in the network, thereby seeking to balance the load for all the wavelengths. This scheme easily splits the long-wavelength paths; hence, only traffic requests that cross a limited number of links will be serviced in the network. In terms of additional overhead communication (e.g., global information is required to measure the least-used

wavelength) LU output is worse than Random. The scheme also requires additional expense for storage and computation; thus, in practice, LU is not favored.

Most-Used (MU)/PACK

MU is the opposite of LU in attempting to find the most widely-used wavelength in the network. It greatly outperforms LU [7]. The overhead of communication, storage, and cost of computation are all close to those in LU. MU also outperforms FF marginally, packing links into fewer wavelengths and retaining the spare power of less-used wavelengths works more.

Min-Product (MP)

In multi-fiber networks Min-Product is used [7]. Min-Product is FF in a single fiber network. MP 's aim is to pack wavelengths into fibers so that the number of fibers in the network is minimized. MP computes first Eq. 11

$$\prod_{l\in\pi(\mathcal{p})} D_{lj} \qquad \textit{Eq. 11}$$

Where $\pi(\mathcal{p})$is the set of ties comprising path $\mathcal{p}$ and matrix $L - by - W$, where D_{lj}indicates the number of fibers assigned to bind l and wavelength j. For wavelengths j, i.e., $1 \leq j \leq W$. If we allow X to denote the set of wavelengths j decreasing the above value, then MP will select the lowest numbered wavelength in X.

Least-Loaded (LL).

Unlike MP, the LL heuristic is designed for multi-fiber networks [20], too. This heuristic selects the wavelength with the greatest residual capacity on the most charged contact along route p. The residual capacity is either 1 or 0 when used in single-fiber networks; thus, the heuristic selects the lowest indexed wavelength with residual capacity 1. Thus, in single-fiber networks, it reduces to FF.

$$\max_{j\in S_{\mathcal{p}}} \min_{l\in\pi(\mathcal{p})} M_l - D_{lj} \qquad \textit{Eq. 12}$$

Where M_l is Number of fibers on link l.

Chapter 4

GNN framework for Routing and Wavelength Assignment (RWA)

There are basically two generic approaches to RWA. The first is to consider RWA as a combined problem consisting of wavelength assignment and routing problem. Generally, this is an NP-hard problem. To reduce the complexity of the problem, it can be decoupled into two separate sub-problems, the routing sub-problem, and the wavelength assignment sub-problem.

In this chapter, we provide a detailed description of the graph neural network framework which is used in routing sub-problem. Then we present information about the graph generation algorithm which is adopted in our work in order to train and test our network. Post-processing is done to make sure that there a path between the pair of nodes is always found, and if the output of the ML framework turns out to be a valid path, it creates a path based on the output of the framework.

Finally, we provide a detailed description of the wavelength assignment algorithms that we have implemented in order to make a comprehensive study.

The framework and other process development are written in the programming Python language, and we use the Networkx library to create network structures. Graph net and Tensorflow libraries are used to implement all learning algorithms.

4.1 Network Routing Workflow (Train Phase)

In this thesis, we exploited the framework proposed in [16]. We use this framework to classify all nodes and edges in the network into two classes: Nodes and edges which are part of the shortest path between two nodes and the nodes and edges that are not in the path.

Figure 12 shows the workflow of the routing process and gives information about the steps we run to have a valid path between the pair of nodes.

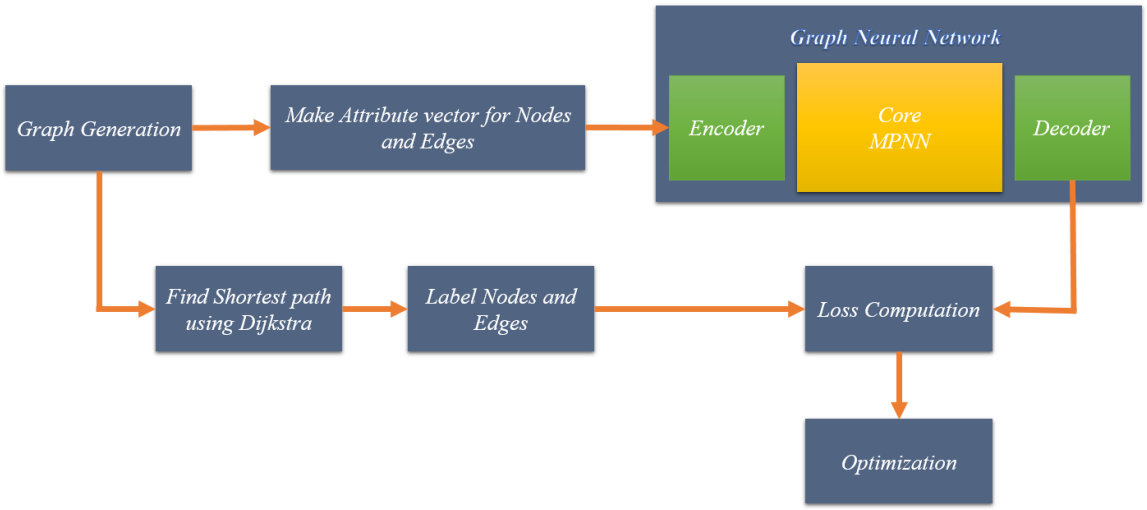


Figure 12: Model training blockdiagram

In order to train the network, since the learning algorithm is supervised, we need target graphs (labeled data) and input graphs (unlabeled data). Target graphs are used in the process of error computation and optimization. And the input graphs are used as part of instruction. The graph is illustrated by attributes of the node and attributes of edge according to the graph definition.

Graph Generation

We used the algorithm proposed in [21]. In this method, two graphs with the same number and position of nodes combine to generate a fully connected graph. The graphs are geographic threshold graphs, but with added edges via a minimum spanning tree algorithm, to ensure all nodes are connected (Figure 13).

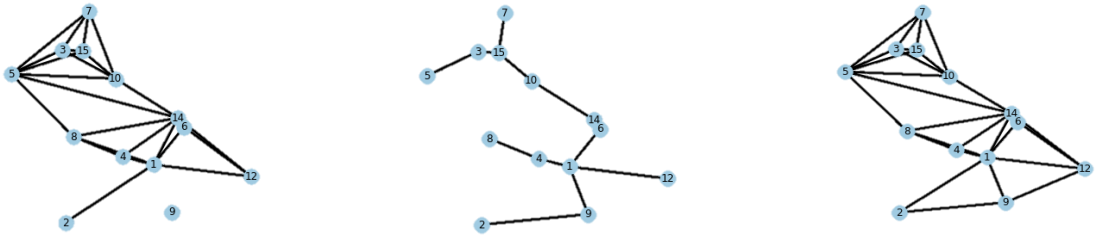


Figure 13. Graph generated by proposed algorithm in [21]. Geographical graph(right) with separated nodes, minimum spanning tree(middle), combined graph(left)

Target graph attribute vector

Dijkstra algorithm for the shortest path problem is used to label the nodes and edges that are part of the shortest path. Distance is used as a weight in the Dijkstra algorithm in order to find the shortest path. It labels each node and edge as following (Figure 13):

The node/edge that belongs to the shortest path is associated with labels [0, 1]. We call them solution nodes and solution edges.

The node/edge that does not belong to the shortest path is associated with labels [1, 0]. We call them non-solution nodes and edges.

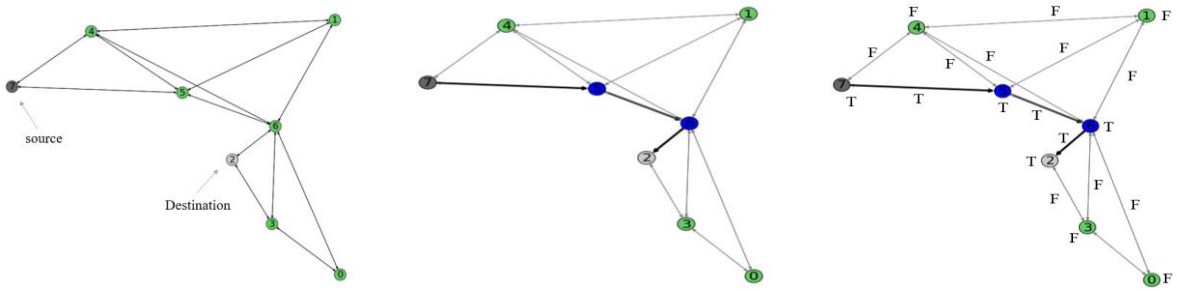


Figure 14: The original graph that the source and destination nodes are marked (Right graph), Routed path graph (middle graph), labeled graph showing the nodes and edges that are part of the path with "T" and others with "F"

The labels of each node are shown in the Table 4. The same is done for attribute vectors. We use this labels as attribute vector.

Table 4: Labeling of nodes in target graph

Nodes ID	Label	Attribute vector
0	False	[1,0]
1	False	[1,0]
2	True	[0,1]
3	False	[1,0]
4	False	[1,0]
5	True	[0,1]
6	True	[0,1]
7	True	[0,1]

Input graph attribute vector

We use input graphs to feed the MPNN. We represent each node and edge in the graph based on its attributes. The attributes of the node in the input graphs consist of a five-element vector that is as:

- Node x-coordinate position

- Node y-coordinate position
- Weight
- Start node
- End node

Weight is an exponential random value that specifies there is a connection between the nodes or not.

The edge attribute is Distance.

ML output

Nodes and Edges in the input graph are labeled at the output of the MPNN. The labels are two-element vectors with different values from 0 and 1 (Table 4). To classify them we interpret the labels and form the output classes. The Eq. 13 is applied in order to define the class.

$$class_{out_i} = argmax\left(Label_{MPNN_i}\right) \qquad \textit{Eq. 13}$$

Table 5: Labeling in MPNN

Nodes ID	Node Labels at the MPNN Output	ArgMax (Label)
0	[6.0417661 -2.23854092]	0
1	[3.35104672 -2.1593072]	0
2	[-4.89505307 3.83677993]	1
3	[5.43695683 -2.35722765]	0
4	[0.72520728 -0.88029439]	0
5	[-0.90006829 0.30690665]	1
6	[-4.14749179 3.46277]	1
7	[-4.68965861 3.7333144]	1

Loss Computation and optimization

The Loss function used is Softmax Cross-Entropy. This softmax function S takes as input a $C - dimensional$ vector z . It outputs a $C - dimensional$ vector y of real between 0 and 1. This function is a normalized exponential and is defined as:

$$y_c = S(z)_c = \frac{e^{z_c}}{\sum_{d=1}^{C} e^{z_d}} \qquad forc = 1 \cdots C \qquad \textit{Eq. 14}$$

The optimization process for the training phase has performed using Tensorflow's ADAM optimizer in order to minimize *softmax cross-entropy* as the classification loss function. The Adam optimization algorithm is an extension of stochastic gradient descent. It is a popular algorithm in the field of deep learning because it achieves good results quickly.

In particular, our GNN configuration in encoder and decoder comprises a fully connected MLP with two hidden layers with 32 neurons and in the core for each message passing step three independent MLP with two hidden layers with 32 neurons perform for edge block, node block, and global block respectively (Figure 15).

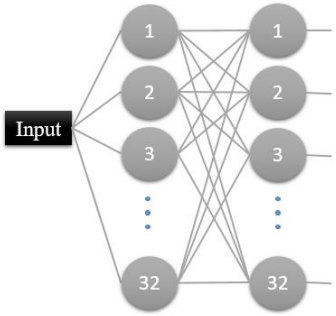


Figure 15. A fully connected MLP with two layers and 32 neurons in each layers

Specifically, the network performs 15,000 training steps (each step uses 32 training graphs as one batch size) with an adaptive learning rate with a starting rate of 0.02. 10 rounds of message passing performed. Such a GNN architecture has been obtained after the number of trial and error experiments until satisfactory generalization results were obtained by checking the classification error in the train, test, and validation sets, along with stable accuracy and loss results. Such a manual inspection process allowed us to identify the best configuration of the parameters of the GNN architecture.

4.2 Network Routing Workflow (Test Phase)

Once the datasets have been generated and the GNN has been trained, in the test phase, we feed the trained network with a graph. The network labels all the nodes and edges at the decoder output. Since node and edges are labeled independently we must ensure the solution constitutes a connected and valid path, so a path checking mechanism is applied. The workflow of the test phase is shown in Figure 16.

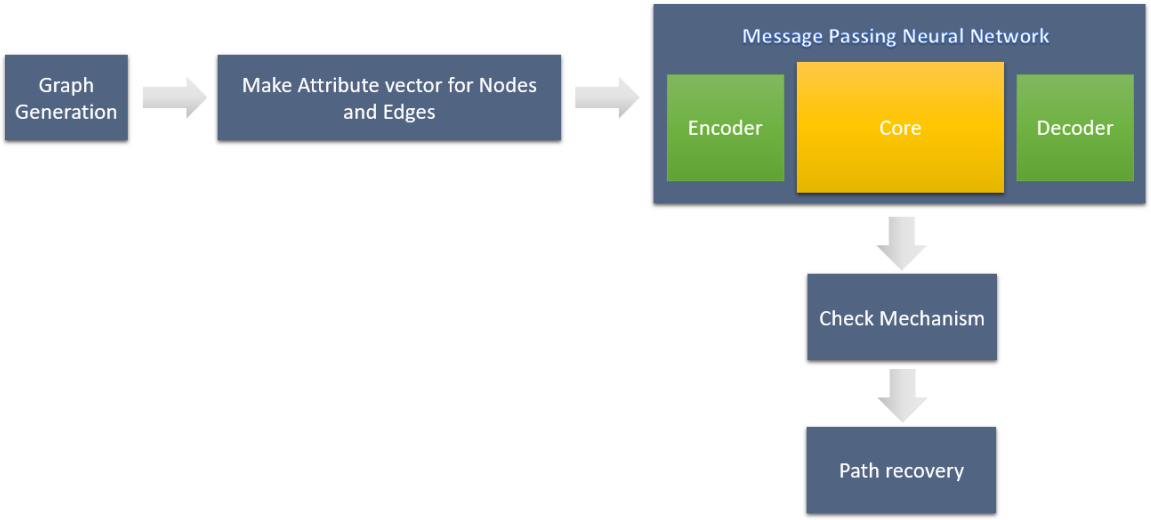


Figure 16: Blockdiagram of proposed algorithm for routing

According to our model, the graph network consists three parts [21] that are shown in Figure 17.

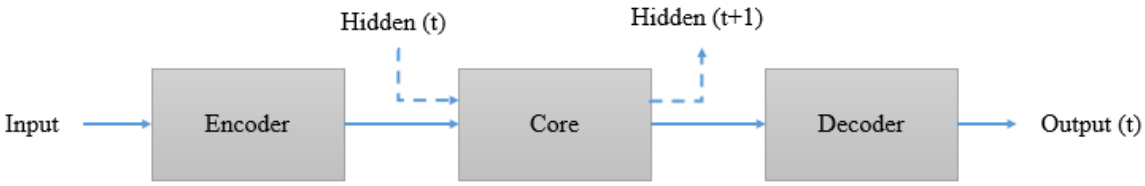


Figure 17. Graph network block

- Encoder: Encodes the edge and node vectors independently. That means two separate MLPs independently map edge and node vectors into a vector with 32-elements.

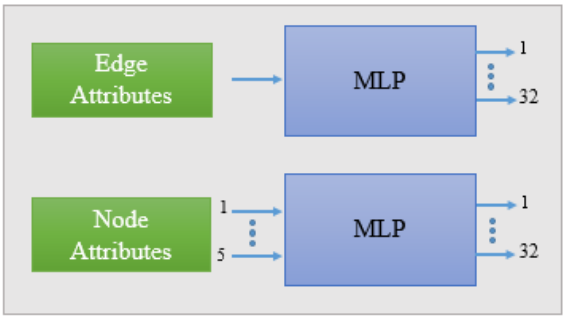


Figure 18. Decoder and Encoder block are using two independent MLPs for edges and nodes of the graph

- Decoder: Decodes the edge and node vectors independently on each message-passing step. So, in the encoder, we have two separate MLPs that independently map edge and node vectors to a 32-elements vector.

- Core: applies massage passing steps N times. The input of core is a concatenation of the core's output at the previous step and the encoder output. The Core block is the Graph Network block which is proposed in [16]. It is consist of three blocks (Figure 17):

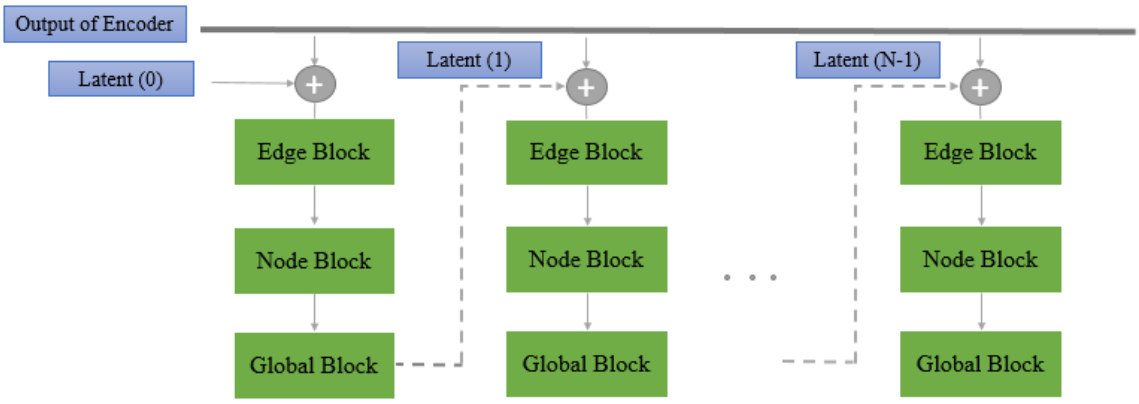


Figure 19: Core block

- Edge Block: Edge features will be updated using an MLP in this block. The MLP input is a concatenation of the preceding of edge features, and of the adjacent node features which are called the sender node and receiver node.
- Node Block: Node features will be updated using an MLP in this block. Firstly, it aggregates the features of adjacent edges. Then a concatenation of the aggregated adjacent edge and the previous node features is used as the input of MLP.
- Global Block: Global features will be updated in this block. Using the MLP a concatenation of aggregated edge features and aggregated node features will be updated.

Check mechanism

At the output of the decoder, we may have an incorrect node or edge labeled as a solution, or the path may be not continuous or be disconnected from source to destination nodes (Figure 20). So we propose a check mechanism over the path composed by node and edge independently labeled to verify whether the solution is a path. We will explain the pseudocode of the mechanism in the following.

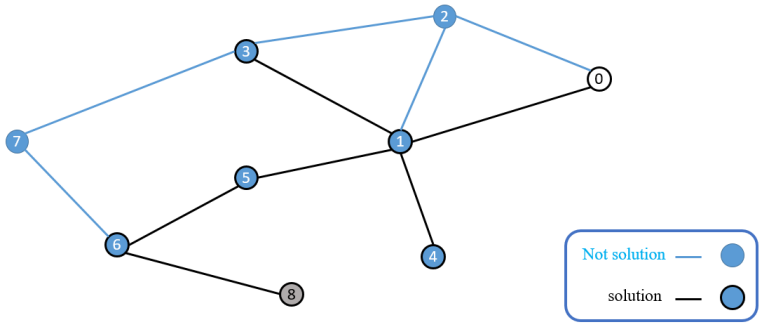


Figure 20 Nodes 3 and 4 are labeled incorrectly, But the path has been obtained.

A graph is defined by $G = (V, E)$, where V is the set of graph nodes and E presents edge set. In the shortest path problem, the aim is finding a path between the source node (v_{src}) and destination node (v_{dst}). The check mechanism starts from v_{src} and traverses the solution labeled nodes, at each step it defines the number of possible outgoing nodes. If the number of outgoing nodes is zero and the current node is the destination node, it supposes that the current node is a dead-end node and removes it from the searching domain. If the number of outgoing nodes is equal to 1, it means that it has just one choice to move, so it selects that node as the next step. If the number is more than one, meaning that there is a multi-way or branch, it adds the node to the branch list and selects one of the possible nodes as the next node. The Branch list helps the algorithm to go back and choose the right way in case of a dead-end or wrong selected path.

The algorithm stops only in the case of reaching destination node or discontinuity of the path.

Algorithm 2. *Check the validity of path using node labels*

$G = (V, E)$
$Branch\ list = []$
$Current\ node = v_{src}$
$While\ path = not\ Found$
    $Num\ nodes, node\ id = OutgoingNodes(G, Current\ node, node\ Labels)$
    $If\ (Num\ nodes = 1)$
        $Next\ node = node\ id$
    $Else\ if (Num\ nodes = 0)$
        $Delete(Current\ position)$
        $Next\ node = Previous\ branch$
            If (branch list = empty)
                break
    $Else\ if (Num\ nodes > 1)$
        $Branch\_list \mathrel{+}= current\_node$
        $Next\ node = random\_node\ from\ available\ nodes$
    $If (Next\ node = v_{des})$
        $Path = Found$
        Current node = $Next\ node$
end

For the edge labels, the same is done but the algorithm traverses solution edges to reach v_{dst} . Then the paths detected using both methods will be combined to get a path with more certainty.

Algorithm 3. *Check the validity of path using edge labels*

```
G = (V, E)
Branch list = []
Current node = v_src
While path = not Found
    Num edges, node id = OutgoingEdges(G, Current node, edge Labels)
    If (Num edges = 1)
        Next node = node id
    Else if (Num edges = 0)
        Delete(Current position)
        Next node = Previous branch
            If (branch list = empty)
                break
    Else if (Num edges > 1)
        Branch_list += current_node
        Next node = random_edge from available edges
    If (Next node = v_des)
        Path = Found
        Current node = Next node
end
```

In the check mechanism one of the following states will be reached.

1. The shortest path will be detected by tracking the node labels or edge labels (Figure 21).

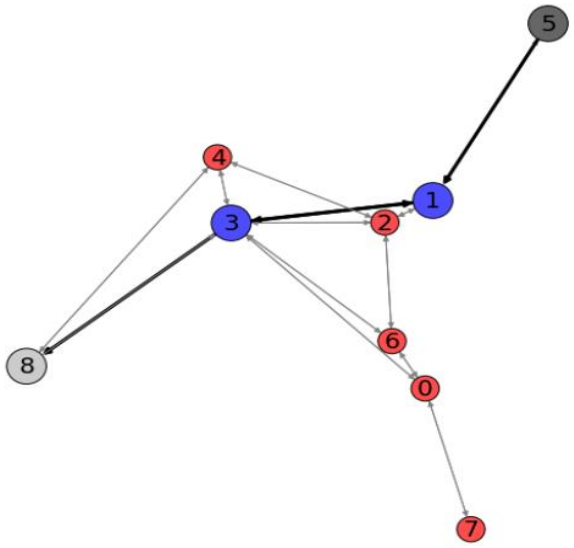


Figure 21: The shortest path is detected by both edge labels and node labels (Large nodes are solution labeled nodes, thicker edges are solution labeled edges).

2. The shortest path will be detected by tracking one of the node/edge labels and will not be detected by the other (Figure 22).

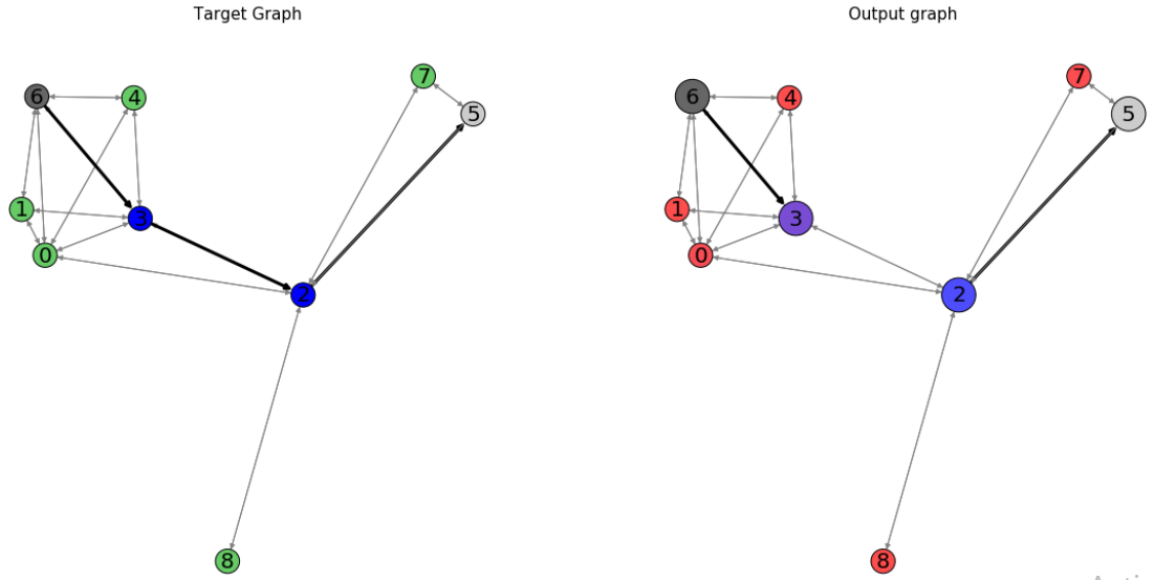


Figure 22: The Shortest Path is detected by tracking node labels, while by following edge labels the shortest path will not be achieved (large nodes are solution labeled nodes, thicker edges are solution labeled edges).

3. The path (not shortest) will be detected by tracking the node labels or edge labels (Figure 23).

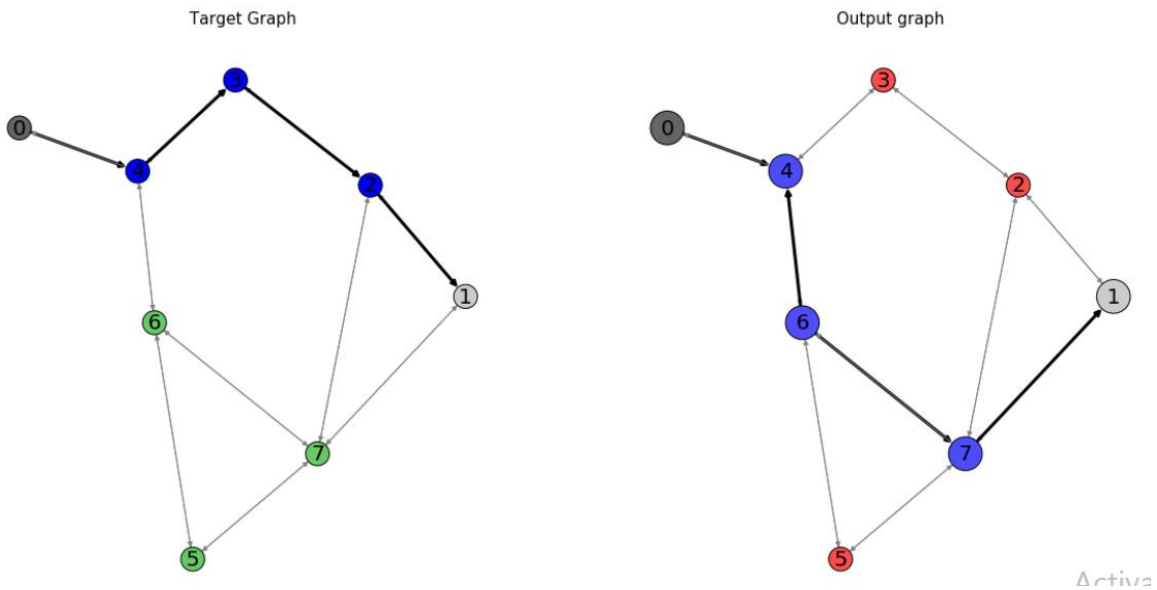


Figure 23: The real shortest path(Left), the path detected by following the node/edge labels which is not the shortest one (Right)

4. The path (not shortest) will be detected by tracking one of the node/edge labels and will not be detected by the other (Figure 24).

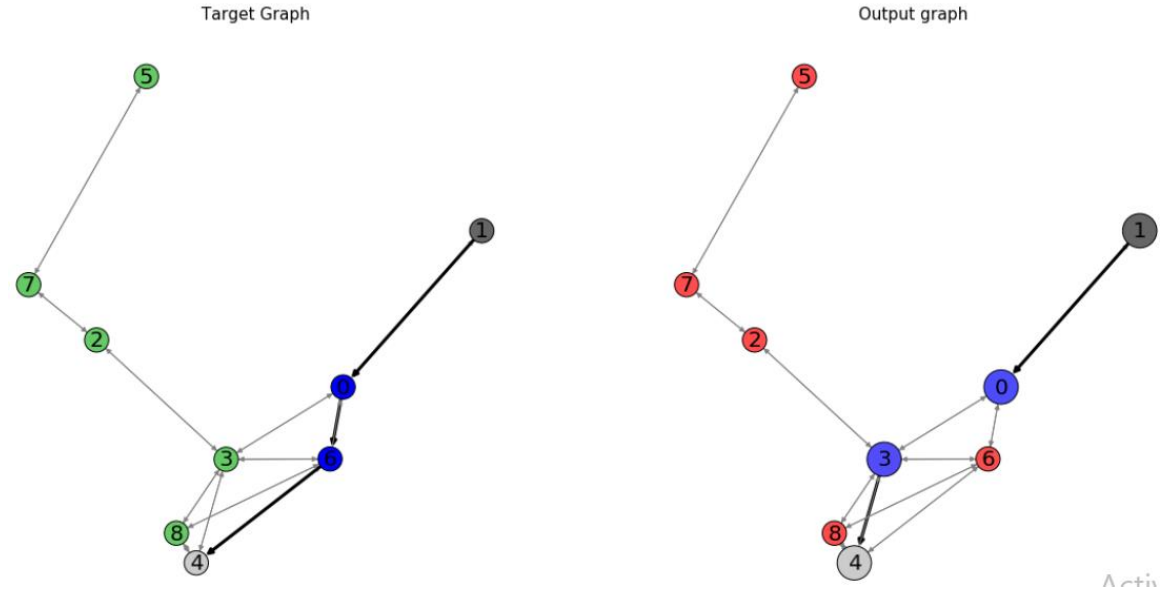


Figure 24: The Path (not shortest) is detected by tracking node labels, while by following edge labels the path will not be achieved (large nodes are solution labeled nodes, thicker edges are solution labeled edges).

5. The path will not be detected (Figure 25).

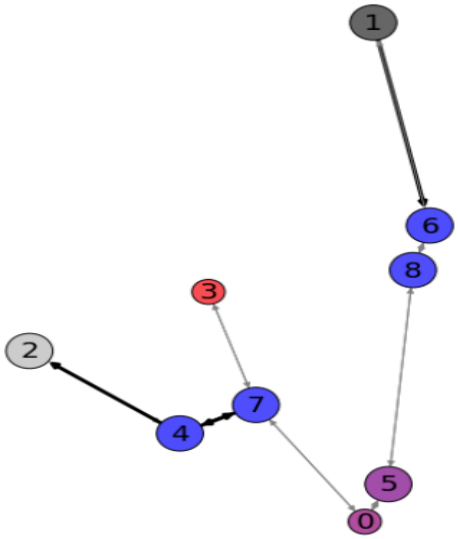


Figure 25: The path is not detected neither by following node labels nor edge labels. (large nodes are solution labeled nodes, thicker edges are solution labeled edges)

Path recovery

When the check algorithms do not succeed in finding a path, we proposed the Path recovery algorithm in order to find a path (Figure 26). The algorithm is implemented as follows:

1. It builds a graph using as link lengths (weights) the labels provided by the neural network as output as follows:
    - If the link is in the solution (label 1) then the link length is 0.
    - If the link is not in the solution (label 0), then the link length is 1.
2. Runs the Dijkstra algorithm on such a graph to find the shortest path between source and destination based on the new weights (Figure 27).

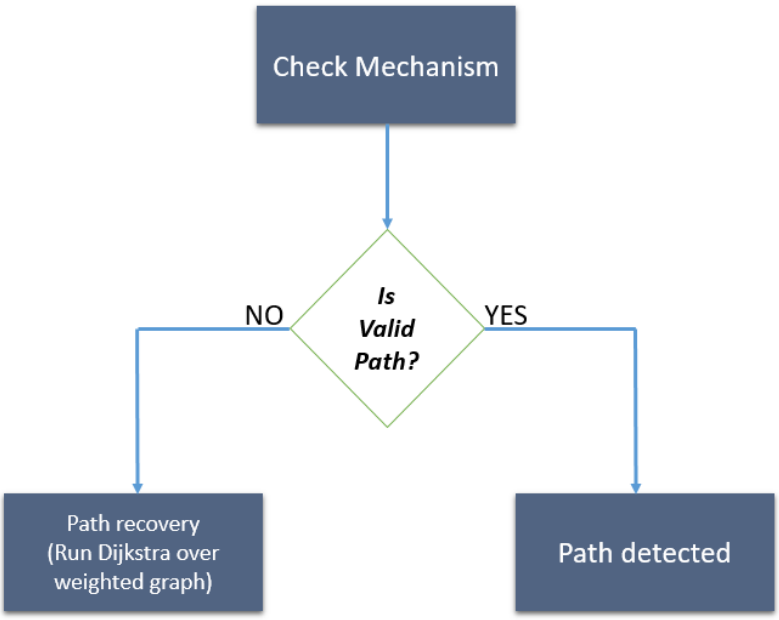


Figure 26: Path recovery

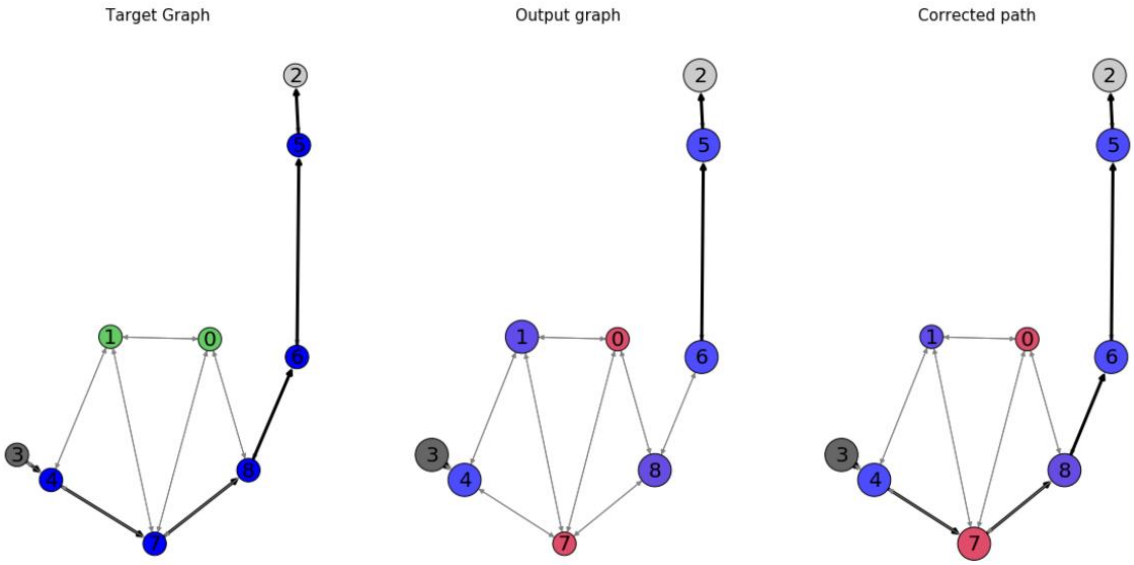


Figure 27: The real shortest path is in left graph, the output labels of our model is in middle image which shows the path has not detected using both node/edge labels. Path is recovered using path recovery algorithm (Right).

4.3 Wavelength Assignment

In our study, we focus on the Dynamic RWA problem and the online algorithms that are proposed in the literature to solve this kind of problem. Also, we assume dynamic path operation in WDM transparent optical networks where each path setup/teardown request will be processed

immediately upon its arrival. No wavelength conversion or signal regeneration is applied (wavelength-continuity constraint).

For the case in which lightpaths arrive one at a time (either incremental or dynamic traffic), heuristic methods must be used to assign wavelengths to lightpaths. For the dynamic problem, instead of attempting to minimize the number of wavelengths as in the static case, we assume that the number of wavelengths is fixed (this is the practical situation), and we attempt to minimize the connection blocking probability [19].

4.3.1 Traffic generation

Traffic requests are generated in accordance with a Poisson process and their holding times follow a negative exponential distribution. Each traffic request consists of a source node (v_{src}), destination nodes (v_{dst}), and one channel to be allocated on one lambda. A lightpath is set up for each connection request as it arrives, and the lightpath is released after some of the holding time of the traffic request.

$$\text{Traffic request} = (v_{src}, v_{dst}, channel) \qquad \textit{Eq. 15}$$

4.3.2 Layered Graph Generation

In order to implement the RWA in optical networks, we use the Dynamic routing algorithm based on the layered–graph model. This method consists e in the following steps:

1. Consider we have an optical network G *(V, E, W)*

   Transform $\boldsymbol{G}$ into a layered graph $G'(V', E')$. Means that we create a copy of the network for each wavelength.

2. Wait for a request.

   If it is a lightpath *connection* request from access Node *s* to access Node *d, s, d* ∈ *V*,

   Go to Step 3,

3. Find a shortest (i.e., cheapest) path p in G from Node s to Node d (e.g., by Dijkstra's algorithm or our method).
4. Find the appropriate wavelength along the path. By checking all the layers and looking at the available capacity of every individual link in the selected path p. If the cost of the path $C_p = \infty$, block the request;

   Otherwise, accept the request and set up the lightpath along the shortest path.

   Update the cost of the intra–layer edges on path *p* to ∞.

5. Hold the path for the connection for the requested hold time, then release the path and set the cost of the intra–layer edges on the to $C_p = 0$.

4.3.3 Implementation of Heuristic Wavelength assignment methods

To evaluate the efficiency of our implemented RWA using GNN, the following experiments has been designed:

We generate requests and try to accommodate them using Random wavelength assignment, first-fit (FF), and Least Used (LU) schemes. In order to route the traffic requests, we exploit our proposed method and the Dijkstra algorithm. Finally, we make a comparison between the aforementioned schemes and routing methods and report the blocking probability for each method.

The way we implemented the experiments is described in the following.

Dijkstra routing, Random Wavelength assignment scheme

According to Figure 28 we generate the traffic requests and route each traffic request using the Dijkstra algorithm in order to have the shortest path. We then assign the wavelength randomly between available wavelengths.

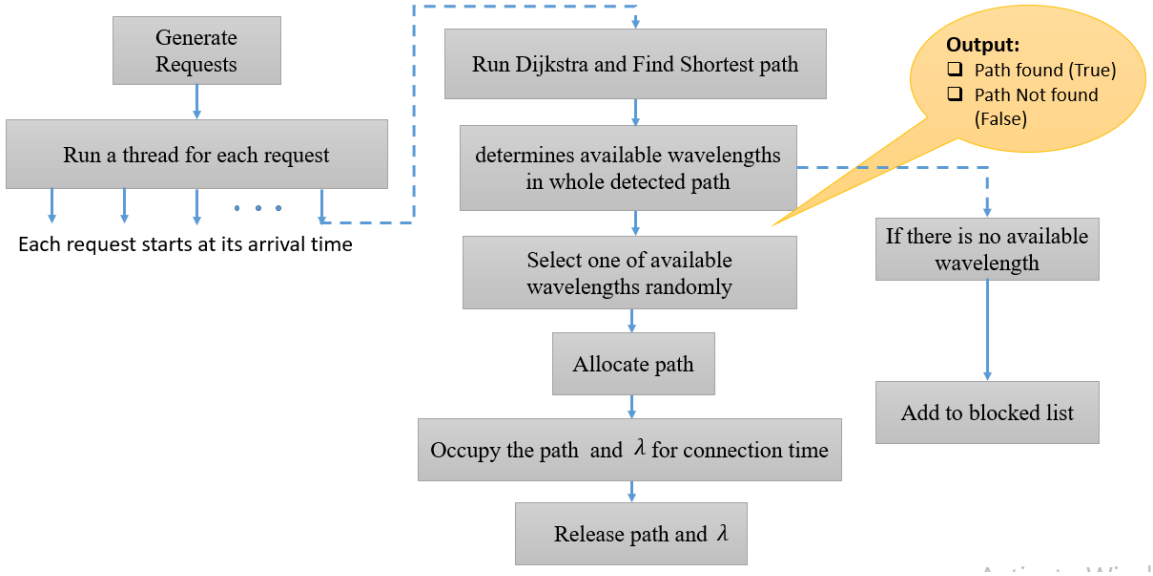


Figure 28: Random WA and Dijkstra routing block diagram

GNN routing, Layered graph, Random Wavelength assignment scheme

To compare the aforementioned method with our proposed method, we implement the following process (Figure 29). In this case, we use the ML-based routing instead of Dijkstra and we used Random wavelength assignment over the layered graph in order to assign an available wavelength to the request [22].

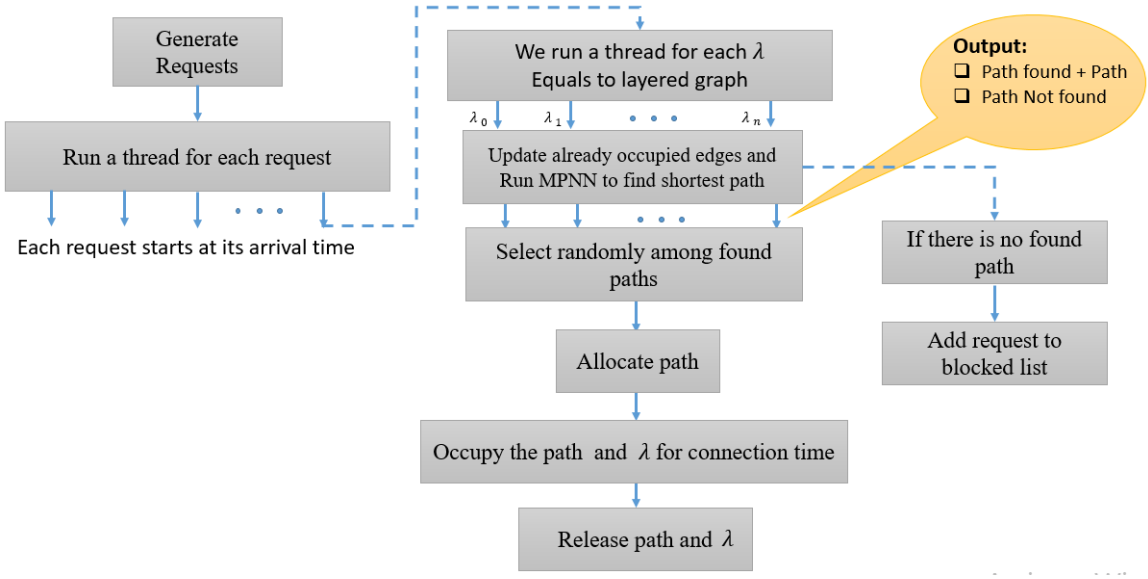


Figure 29: GNN routing, Layered graph, Random Wavelength assignment block diagram

Dijkstra routing, First-Fit Wavelength assignment scheme

In this experiment, to route a path between sender and receiver nodes we have exploited Dijkstra and used the FF wavelength assignment method (Figure 30).

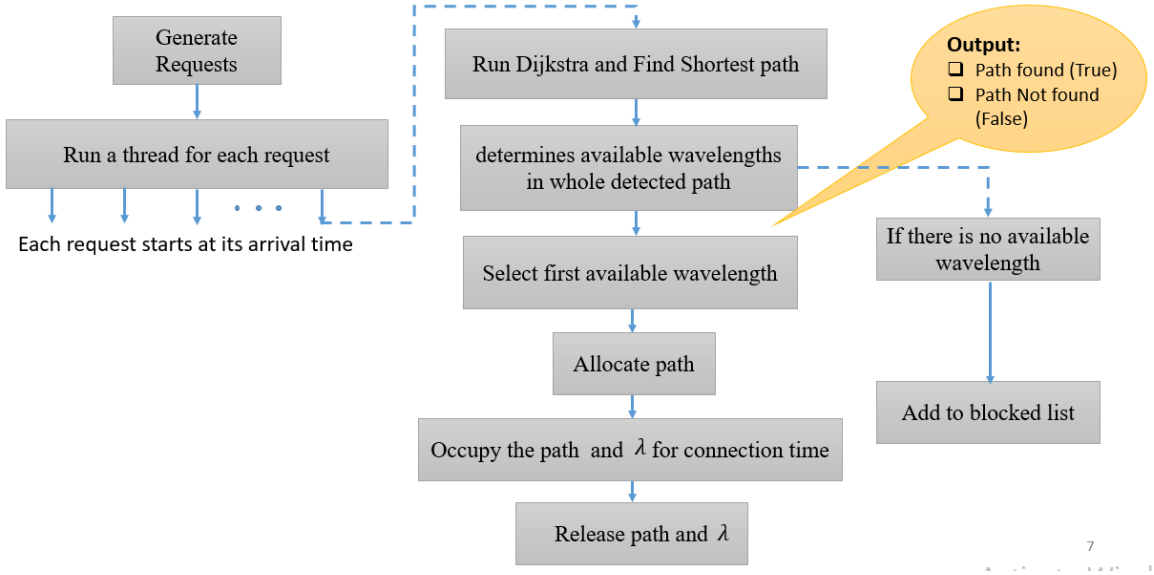


Figure 30: First-fit WA and Dijkstra routing block diagram

GNN routing, Layered graph, First-Fit Wavelength assignment scheme

In This experiment, to obtain first-fit approach, we add to the link weights a term $lambda * epsilon$ (normally the link weights are distances), where epsilon is a small quantity (e.g. 0.1) and

lambda is the index of the considered wavelength (i.e., the leftmost wavelength has lambda=0, the first one has lambda =1 etc.). This way, the leftmost wavelengths should be preferentially used.

If a feasible assignment is found, allocate the traffic request, otherwise, the request is blocked.

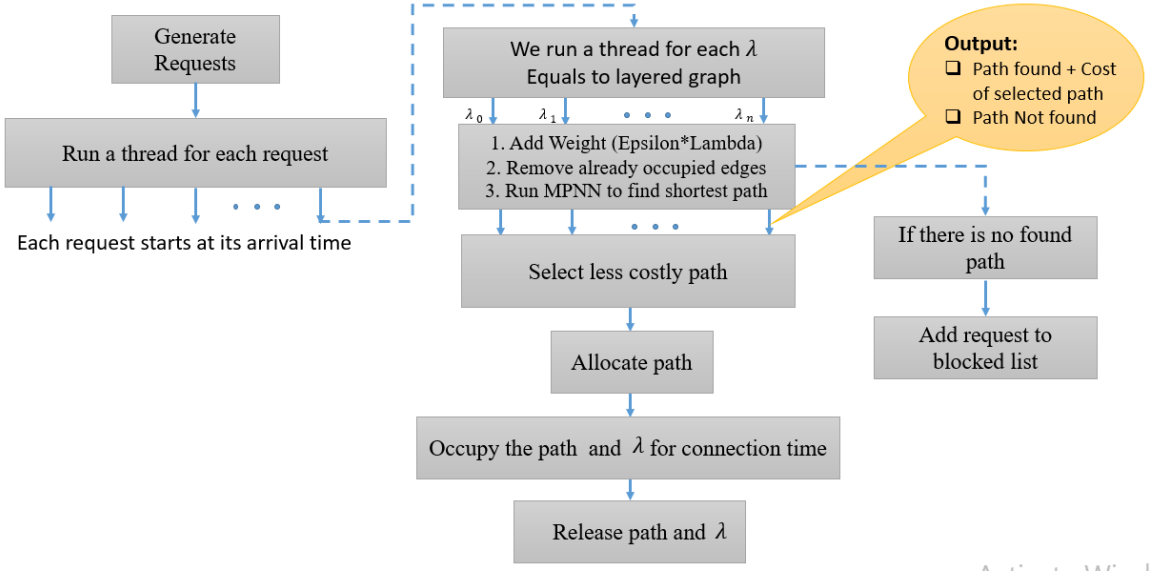


Figure 31: GNN routing, Layered graph, First-fit Wavelength assignment block diagram

Dijkstra routing, Least Used Wavelength assignment scheme

In this experiment, to route a path between sender and receiver nodes we have exploited Dijkstra and used LU wavelength assignment method (Figure 32). In the LU we choose the wavelength that the least used in the network.

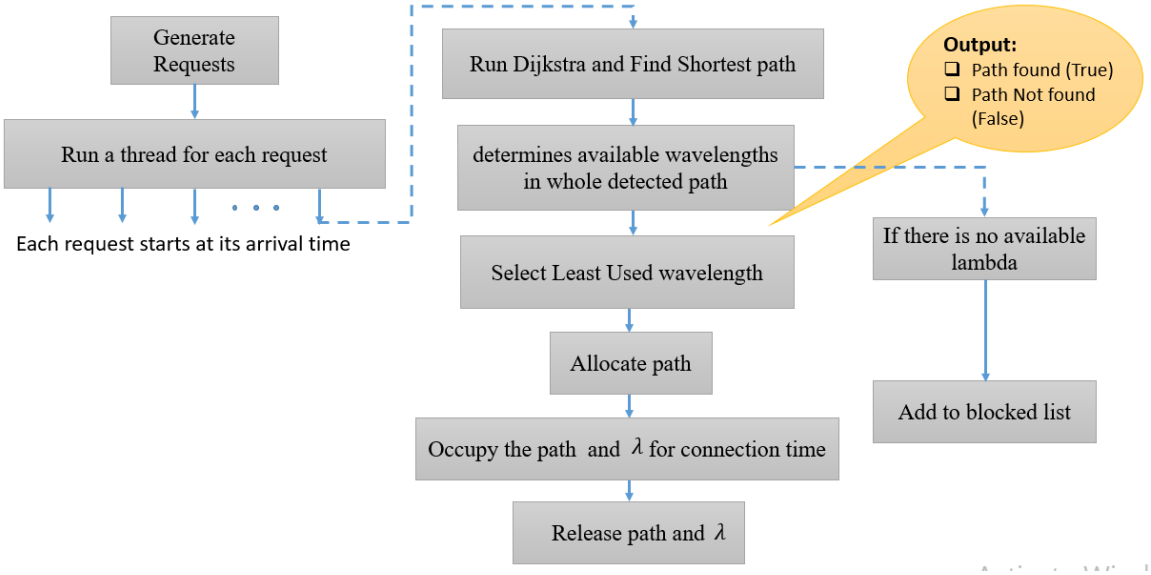


Figure 32: Least-used WA and Dijkstra routing block diagram

GNN routing, Layered graph, Least-Used Wavelength assignment scheme

In this experiment, the RWA is done using GNN and Least Used scheme over layered graph (Figure 33).

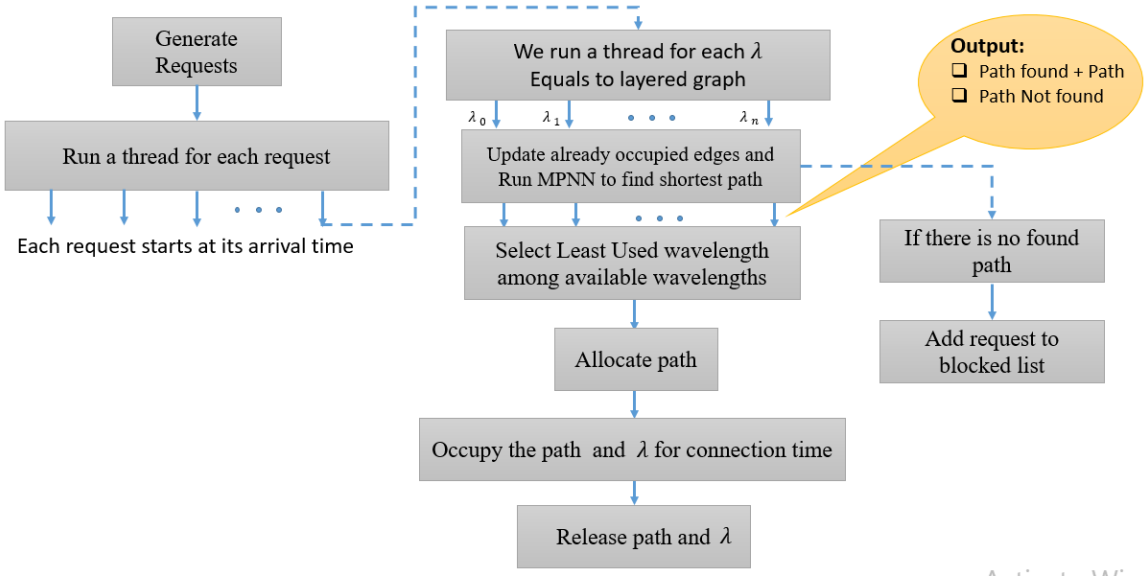


Figure 33: GNN routing, Layered graph, Least-used Wavelength assignment block diagram

4.4 Performance evaluation metrics

The last subsection is characterized by the description of the metrics involved in the routing accuracy measurements (node and edge classification problem) and the explanation of the metrics that are used in the evaluation of wavelength assignment algorithms.

4.4.1 Routing performance evaluation

Our routing solution is divided into two subsections, first is the evaluation of the performance of the node/edge classifier. Second related to the path detection algorithm.

Node/edge classifier performance metrics are as following.

STR (training fraction examples solved correctly):

$$STR = \frac{Number\ of\ correctly\ solved\ examples\ in\ training\ dataset}{Number\ of\ total\ examples\ in\ training\ dataset} \qquad Eq.\ 16$$

STR is the `float` fraction of training graphs that are completely correctly labeled.

SGE (test/generalization fraction examples solved correctly):

$$SGE = \frac{Number\ of\ correctly\ solved\ examples\ in\ test\ dataset}{Number\ of\ total\ examples\ in\ test\ dataset} \quad \textit{Eq. 17}$$

SGE is the `float` fraction of test/generalization graphs that are completely correctly labeled.

CTR (training fraction of nodes/edges labeled correctly):

$$CTR = \frac{Number\ of\ correctly\ labeled\ nodes/edges\ in\ training\ dataset}{Total\ Number\ of\ nodes/edges\ in\ training\ dataset} \quad \textit{Eq. 18}$$

CTR is the `float` fraction of correctly labeled nodes/edges in training dataset.

CGE (test fraction of nodes/edges labeled correctly):

$$CGE = \frac{Number\ of\ correctly\ labeled\ nodes/edges\ in\ test\ dataset}{Total\ Number\ of\ nodes/edges\ in\ test\ dataset} \quad \textit{Eq. 19}$$

CGE is the `float` fraction of correctly labeled nodes/edges in test dataset.

SG (fraction of graphs solved correctly):

To evaluate the path detection algorithm, we define the Solved Graph (SG) metric:
This metric is the `float` fraction of correctly solved graphs. The correctly solved graphs are the graphs where a feasible path between pair of sender and receiver nodes has detected.

$$SG = \frac{Number\ of\ \text{correctly solved graphs}}{Total\ Number\ of\ tested\ graphs} \quad \textit{Eq. 20}$$

4.4.2 Wavelength assignment performance evaluation

Consider a WDM optical network architecture in which the lightpaths between the access nodes are set up *dynamically, i.e.,* on-demand. Whenever a new lightpath is requested, the dynamic routing scheme either selects an available path, or it blocks the request if no such path can be found. However, if multiple paths exist, the "best" path that is likely to minimize blocking in the future should be selected. Unfortunately, finding the "best" path is not straightforward. In fact, it has been proved to be an NP-complete problem. So to evaluate the wavelength assignment algorithm we use the Blocking Probability metric.

$$Blocking\ Probability = \frac{Number\ of\ refused\ pathlight\ requests}{Total\ Number\ of\ pathlight\ requests} * 100 \qquad \textit{Eq. 21}$$

Chapter 5

Results

In this chapter, both the results of the routing algorithm and wavelength assignment technique are reported. In the first section, the results obtained in the routing algorithm on the variety of network topologies are shown. The second section reports the results achieved by different wavelength assignment techniques which we have implemented in this study.

5.1 ML Based Routing Results

In this section, we first look at how dataset graphs are generated and give a complete description of the dataset. Then we report the performance of the proposed method for the routing problem.

5.1.1 Dataset Generation

We perform experiments on synthetic datasets. In the database, graphs are a superposition of geographical threshold graphs and minimum spanning tree graphs.

The geographical threshold graph model allocates n nodes uniformly at random in a rectangular domain. Every node u is assigned a weight w_u. An edge connects two nodes u and v if:

$$w_u + w_v \geq \theta r^{\{\alpha\}} \qquad \textit{Eq. 22}$$

Where r is the Euclidean distance between u and v, and θ, α are parameters.

Then we compute a minimum spanning tree over the nodes. A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected graph that connects all the nodes together, without any cycles and with the minimum possible total edge weight.

In the case of the geographical threshold graph, we may have some graphs with disconnected portions, so in order to avoid the presence of disjoint nodes and having connected graphs, we combine these two graphs.

5.1.2 Dataset Statistics

In this study, we train our GNN with different sizes of network topology and we test the GNN in order to evaluate the scalability and the ability of generalization of our method.

We train the network with graph topologies with 10~200 nodes and test with graphs with 10~400 nodes.

5.1.2 GNN Hyperparameters

To optimize the hyperparameter, a grid search has been used and the optimal parameters have been obtained by searching exhaustively through a specified subset of hyperparameters. Optimized parameters are as following:

- Number of Message Passing Steps
- Number of Iterations
- Learning rate
- Number of hidden Layers
- Number of Neurons in each layer

To analyze the parameters, we used the same set of graph topologies to make the results comparable.

Change of the Number of Message Passing Steps

Here we changed the Number of Message Passing Steps (Table 6) in the framework and we analyzed the effect of changing this parameter over the GNN performance.

Table 6: Change the MP steps

Iterations	***Message Passing steps***	***Learning Rate***	***Number of Layers***	***Number of Neurons***	***Time (Hours)***	***STR***
10000	8	1e-3	2	16	0.58	0.9233
10000	10	1e-3	2	16	0.67	0.9433
10000	12	1e-3	2	16	0.73	0.9437
10000	14	1e-3	2	16	0.79	0.9509

Change the Learning rate

Table 7: Change the learning rate

Iterations	***Message Passing steps***	***Learning Rate***	***Number of Layers***	***Number of Neurons***	***Time (Hours)***	***STR***
10000	10	1e-1	2	16	1.04	0.8333
10000	10	1e-2	2	16	0.79	0.9397
10000	10	1e-3	2	16	0.67	0.9433

Change the Number of hidden Layers and neurons in each layer

Table 8: Change the number of neurons

Iterations	***Message Passing steps***	***Learning Rate***	***Number of Layers***	***Number of Neurons***	***Time (Hours)***	***STR***
10000	10	1e-3	2	16	0.67	0.9433
10000	10	1e-3	2	32	0.9	0.9570
10000	10	1e-3	2	64	1.7	0.9627
10000	10	1e-3	3	16	0.78	0.9085
10000	10	1e-3	3	32	0.99	0.9310
10000	10	1e-3	3	64	1.89	0.9528

Adaptive learning rate (LR)

In this test, we assume the same values for all the parameters in the framework, and we used both fixed and adaptive learning rate. Two tables (Table 9, Table 10) show that adaptive LR instead of fixed LR has a better STR factor.

	Table 9: Fixed and Adaptive LR with 2 hidden layers					
Iterations	***Message Passing steps***	***Learning Rate***	***Number of Layers***	***Number of Neurons***	***Time in Hours***	***STR***
10000	10	Fixed	2	32	0.9	0.9570
10000	10	Adaptive	2	32	0.93	0.9840

	Table 10: Fixed and Adaptive LR with 3 hidden layers					
Iterations	***Message Passing steps***	***Learning Rate***	***Number of Layers***	***Number of Neurons***	***Time (Hours)***	***STR***
10000	10	Fixed	3	32	0.99	0.9310
10000	10	Adaptive	3	32	1.0	0.9632

By tuning the above parameters, the following settings achieved.
We used 10 rounds of message passing, Adaptive learning rate, 10000 for the iterations, Although the model with 3 hidden layers has shown slightly better results in terms of factor STR, it significantly increases the time complexity in the model, so we used two hidden layers and 32 neurons in each layer in the model configuration for further experiments.

Node/Edge classifier performance evaluation

To evaluate the performance of the GNN classifier and routing algorithm we have conducted different experiments.

Experiment 1

In the first experiment, we have trained the classifier with graphs with 10~15 number of nodes. After every 20 iterations (total number of iterations is 10000), we run the validation process and feed the classifier with the same size graphs, then we compute the performance metrics. These metrics will be presented in the following.

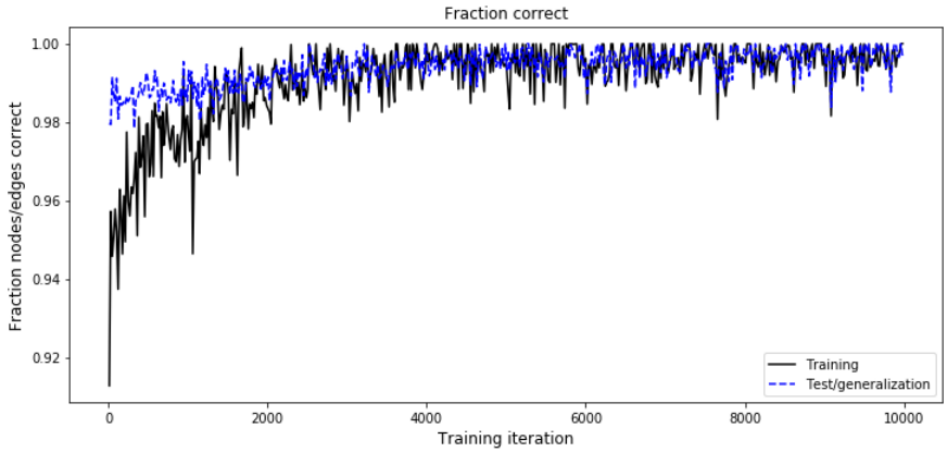


Figure 34: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp1.

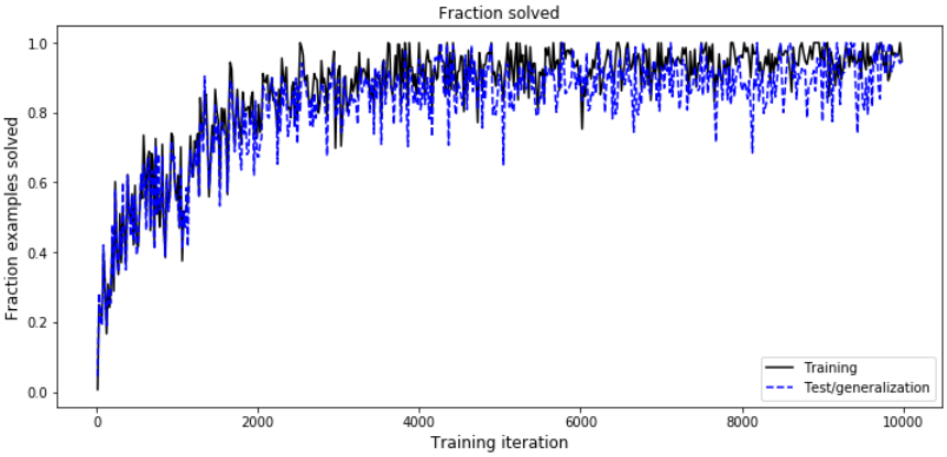


Figure 35: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp1.

In Figure 34, it is shown that only a few labels are incorrect at the end of the training. Remember that, this metric does not capture whether or not the shortest path is found and we consider both solution and non-solution labels. Because the majority of nodes/edges are correctly labeled as non-solution in this metric, the value closely approaches to 1.

Figure 35 shows that after training, the classifier has correctly classified nodes/edges labels and 98% of the shortest paths are identified correctly.

Experiment 2

Train with Graphs with 10~15 number of nodes.

Validation with Graphs with 15~30 number of nodes,

In this experiment we evaluate how well the models generalize to graphs which are up to twice as large as those on which they were trained.

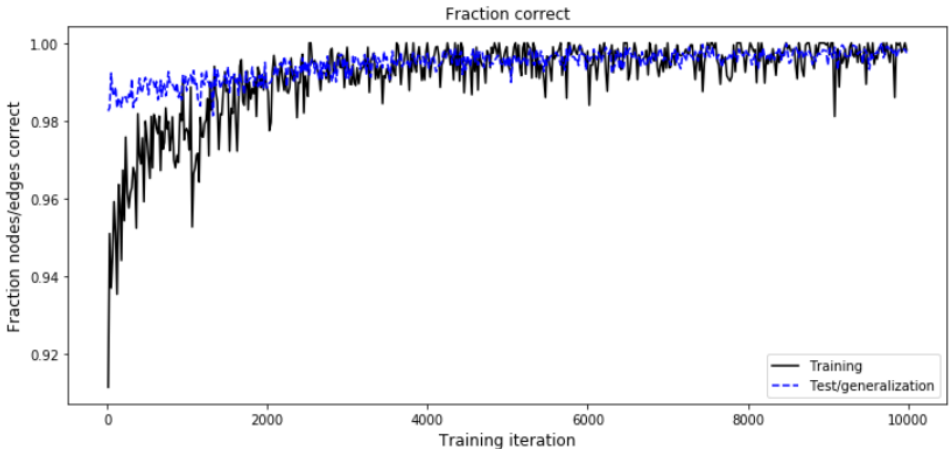


Figure 36: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp2.

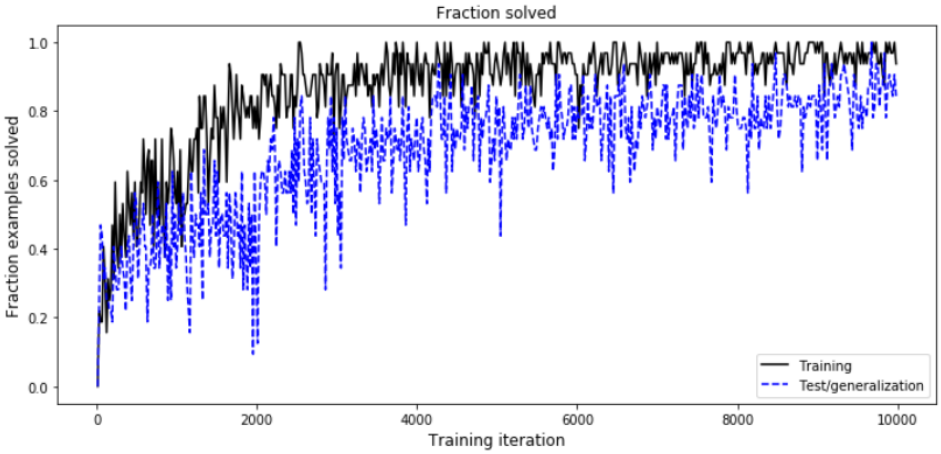


Figure 37: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp2.

The Figure 36 shows that the classifier in the training phase could correctly classify almost all of the nodes/edges in the training graphs at the last iterations and about 90% of the validation graphs (Figure 37).

Experiment 3

Train with Graphs with 30~40 number of nodes.

Validation with Graphs with 30~40 number of nodes,

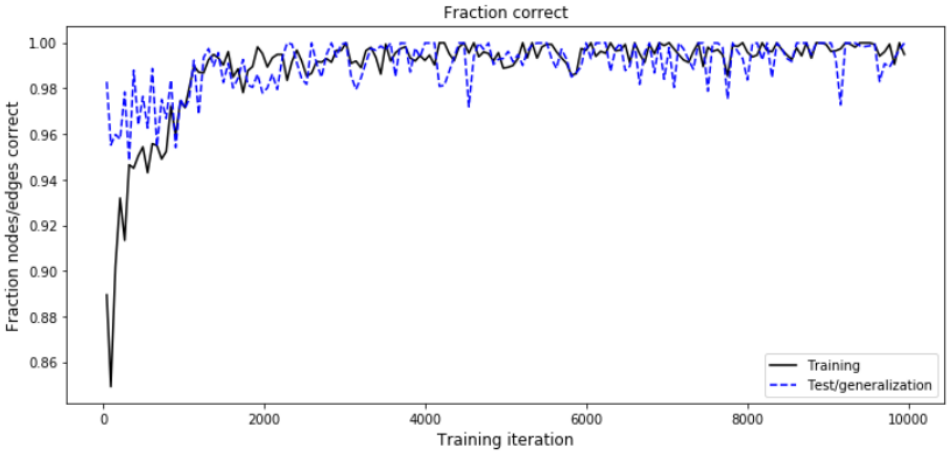


Figure 38: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp3.

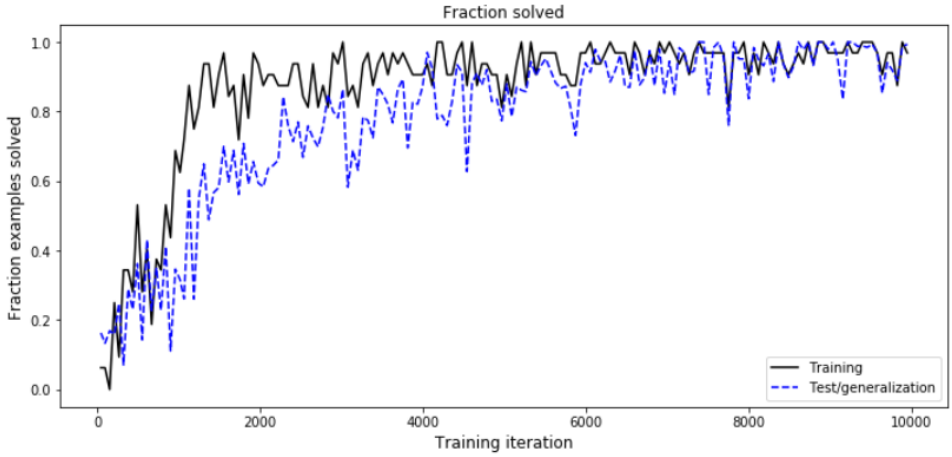


Figure 39: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp3.

In the Figure 38, the classifier in the training phase could correctly classify almost all of the nodes/edges in the training graphs. In Figure 39, in the last iterations the classifier's accuracy reaches to about 98% of the train/validation graphs.

Experiment 4

Train with Graphs with 30~40 number of nodes.

Validation with Graphs with 60~80 number of nodes,

In this experiment, we also evaluate how well the models generalize to graphs which are up to twice as large as those on which it was trained.

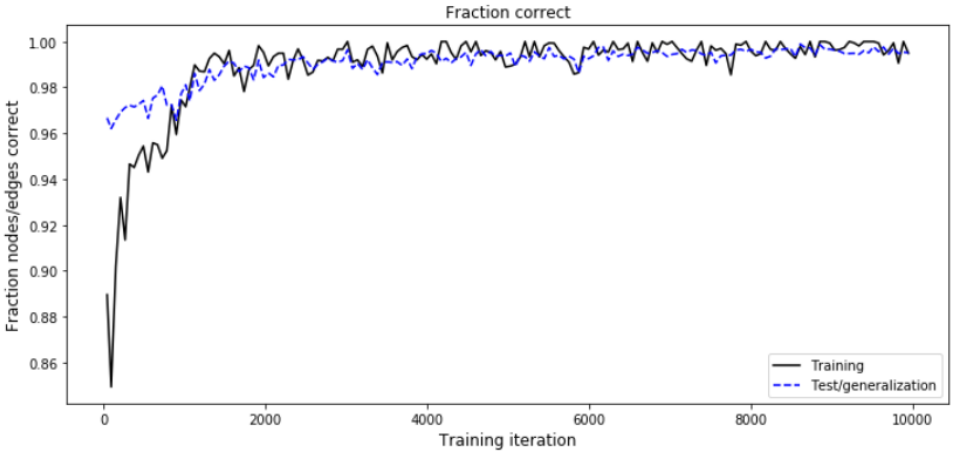


Figure 40: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp4.

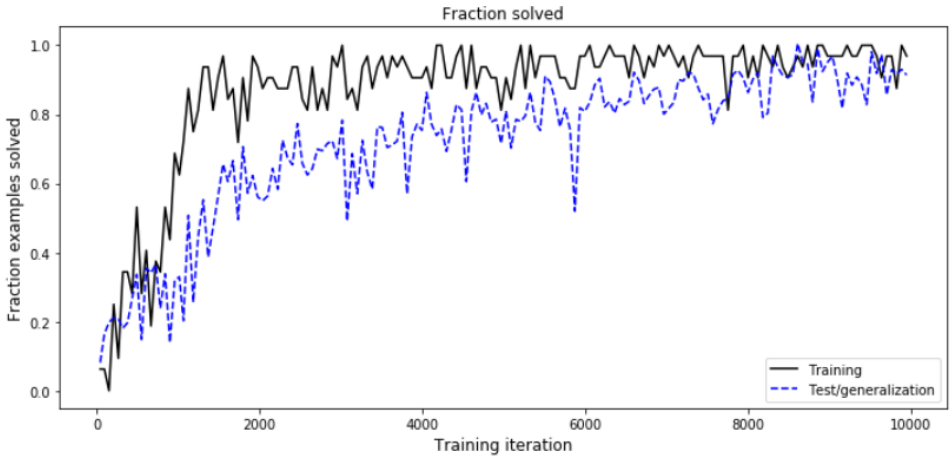


Figure 41: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp4.

This experiment shows the model can be generalized for the graphs which are larger than the graphs that were it has seen during training time.

Experiment 5

Train with Graphs with 120~150 number of nodes.

Validation with Graphs with 120~150 number of nodes,

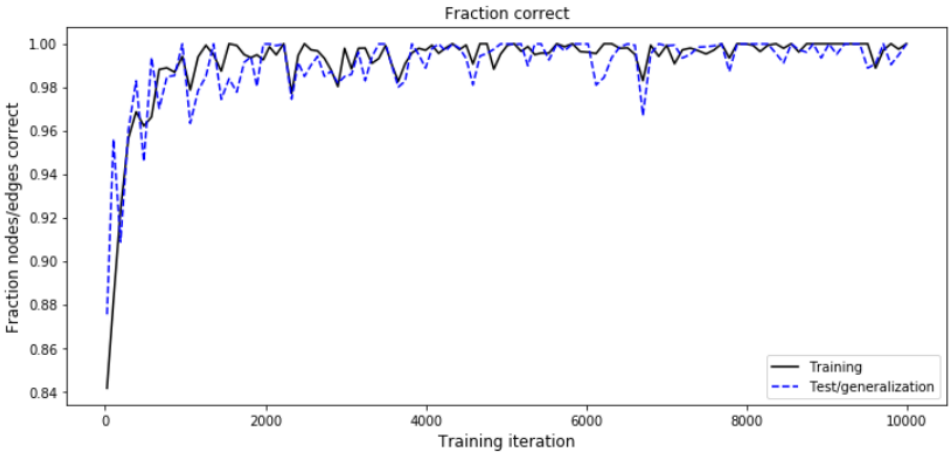


Figure 42: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp5.

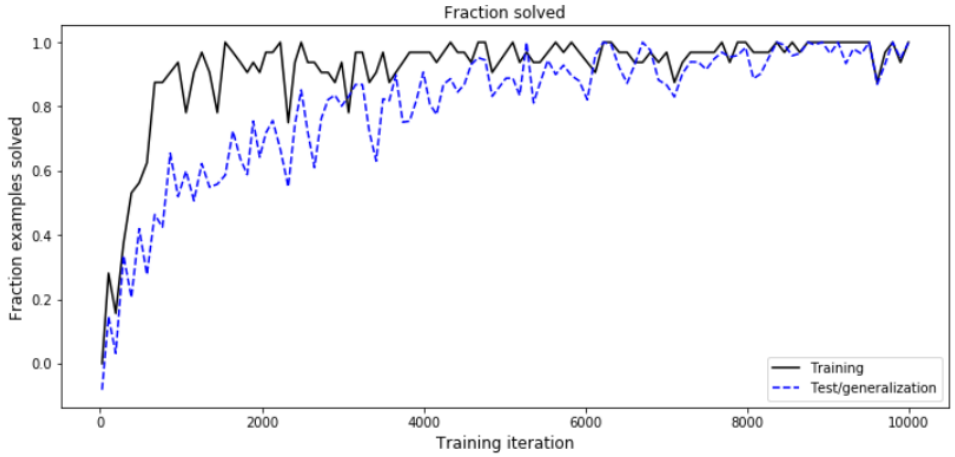


Figure 43: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp5.

In the Figure 42, the classifier in the training phase could correctly classify almost all of the nodes/edges in the training graphs. In Figure 43, in the last iterations the classifier's accuracy reaches to about 98% of the train/validation graphs.

Experiment 6

Train with Graphs with 120~150 number of nodes.

Validation with Graphs with 240~300 number of nodes,

In this experiment, we also evaluate how well the models generalize to graphs which are up to twice as large as those on which it was trained.

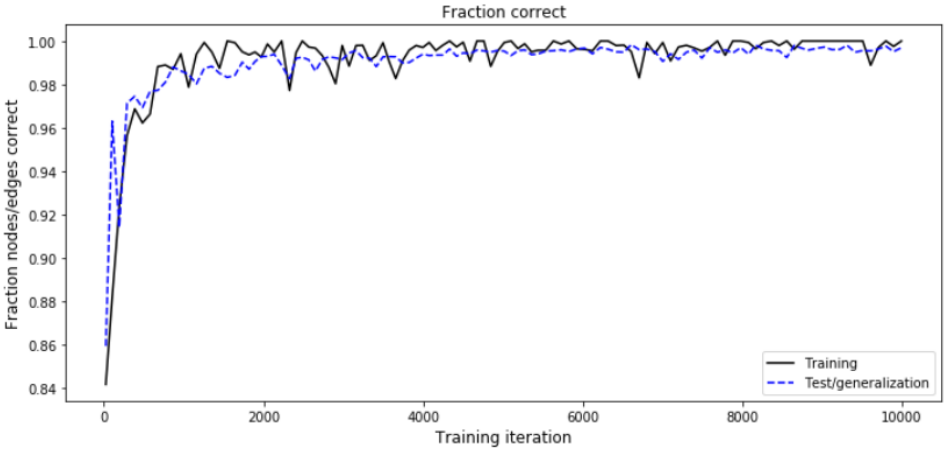


Figure 44: Fraction of correct labels (CTR/CGE) during train phase and validation test in Exp6.

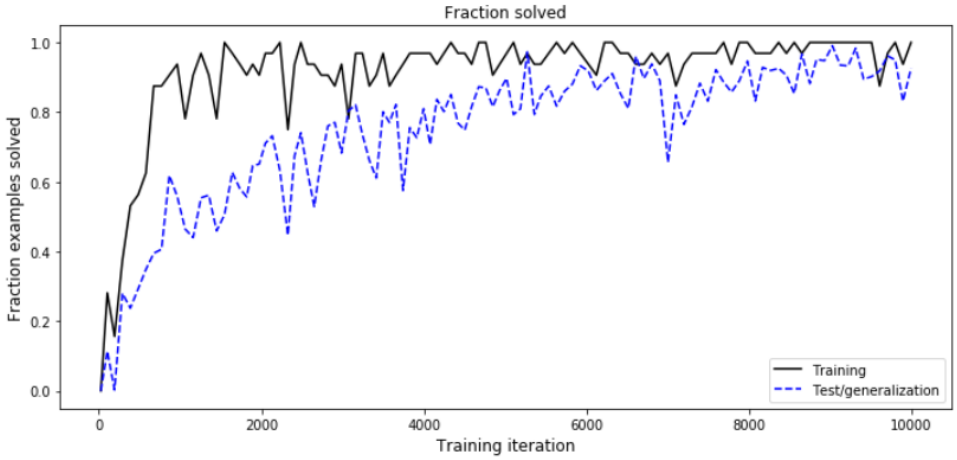


Figure 45: Fraction of solved graphs (STR/SGE) during train phase and validation test in Exp6.

According to the results, it can be concluded that the classifier has performed well in identifying the label of nodes and edges in train dataset graphs and the shortest path has not been found except for a few graphs in the database. However, in graphs where all of the nodes/edges have not labeled correctly and we have reported them as not solved graphs, it is possible that solution node labels or edge labels indicate the shortest path or solution node/edges may provide a valid path between the sender and receiver, which we will examine in the next section.

Routing performance evaluation

In this section, we used the models which we had trained in the previous section, to define the detected path performance we run the check mechanism over the output of the classifier. We do some experiments over the models which are trained in the previous sub-section and the SG metric will be reported. This metric shows the ability of our method in the case of routing.

To have reliable results, we generate a test dataset containing 1000 graphs with different sizes as follows.

Test 1

In this experiment, we have leveraged the trained network in the Experiment 1 and we generate 1000 graphs, each graph has 15 ~ 30 number of nodes.

SG = 1.0

This experiment shows that the framework can solve the shortest path problem for the graphs which are two times bigger than the graphs we used in the training phase.

According to Figure 37, we can see that the network solves about 90% of all tested graphs with sizes bigger than training graphs, But here we get 100% solved graphs. It shows that even if we cannot solve the problem by node labels, the problem can be solved by tracking edge labels or vice versa.

Test 2

In this experiment, we have leveraged the trained network in Experiment 3 and we generate 1000 graphs, each graph has 60 ~ 80 number of nodes.

SG = 1.0

Test 3

In this experiment, we have leveraged the trained network in Experiment 5 and we generate 1000 graphs, each graph has 300 ~ 400 number of nodes.

SG = 0.998

The obtained value of SG shows that the model could find the path in 998 graphs out of 1000 input graphs.

As we can see, in some cases that the shortest path has not been correctly detected by the model. A valid path (not shortest) is detected by edge labels or node labels and just about less than one percent of the whole test dataset that the path has not achieved. For these cases we run the Path recovery to get a valid path, this algorithm uses Dijkstra [6] and guarantees the path.

5.2 Wavelength assignment numerical results

In this section performance of the dynamic routing algorithm has been studied via simulations. The simulations are conducted on randomly generated networks of different sizes. For each network configuration, the simulations results are averaged over ten random network topologies. Lightpath requests are assumed to arrive according to a Poisson process with mean arrival rate r, this rate has been set in our following experiment equal to 10. Holding times of the lightpaths are exponentially distributed with the $\mu = 3.0$. The probability distribution of the arrival time and the holding time of the lightpath requests have been shown in Figure 46 and Figure 47 respectively. Also, once a lightpath request is blocked, it is removed from the lightpath – request queue. The

traffic pattern is assumed to be uniform, that is, the lightpath requests are uniformly distributed over all the access node pairs.

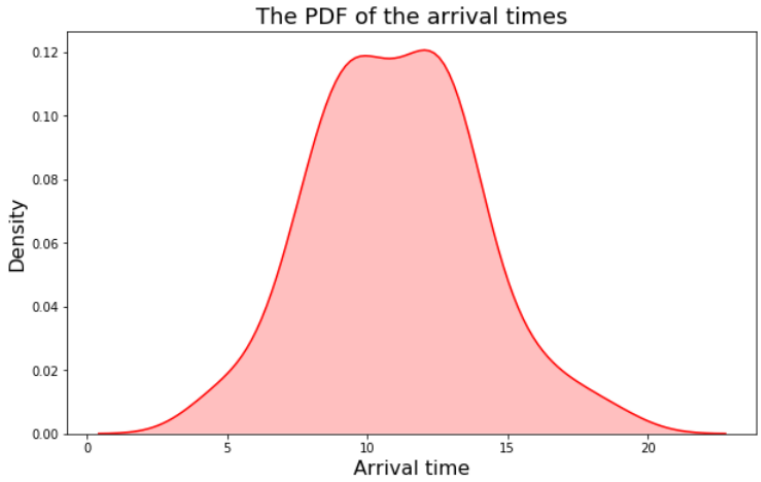


Figure 46: PDF of arrival times of the lightpaths

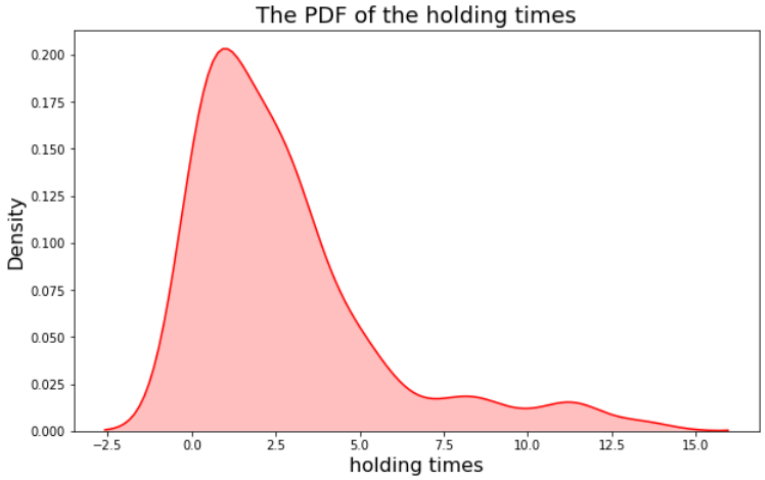


Figure 47: PDF of holding times of the lightpaths

To evaluate the performance of the implemented algorithms which are explained in previous chapter, we performed the following experiment.

In this experiment, a graph with 100 nodes is generated, the number of wavelengths is set to 8. In each run of the experiment, a number of requests with the aforementioned specifications have been generated. Each lightpath request asks for a route from a sender node to a receiver node over a wavelength. Once a request arrives, either it will be accommodated in the graph by the wavelength assignment algorithm or it will be blocked. We analyze the blocking probability related to each wavelength assignment algorithm by a different number of lightpath requests (Figure 48).

In order to accommodate lightpath requests, we have exploited two different methods for routing in addition to random (R), First-fit (FF), and Least-used (LU) heuristic methods for wavelength assignment. In our suggested method we have exploited layer-graph model.

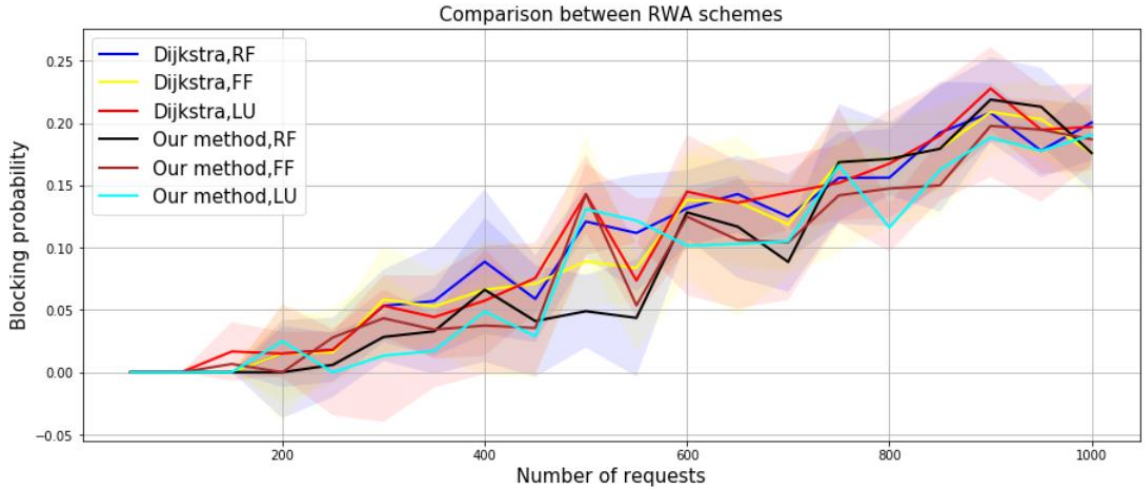


Figure 48: Blocking probability against number of requests

The performance of 100–node graphs with 8 number of wavelengths per link is shown in Figure 48. The figure shows the average blocking probability of the lightpath requests as a function of the number of arriving lightpath requests.

According to the results, we obtain that the layer graph model can improve the blocking rate slightly. For example, when the number of lightpath requests is 700, blocking probability in the set of lightpath requests that served by our method is lower than the other methods. The reason for the decrease in the blocking probability is that whenever a request arrives we may have different paths in each layer and if a route between the sender node and receiver node is busy, the algorithm tries to find another path. So this reduces the blocking probability in our method.

Chapter 6

Conclusion

Recently, much attention has been devoted to the question of whether/when traditional network protocol design, which relies on the application of algorithmic insights by human experts, can be replaced by a data-driven (i.e., machine learning) approach. We explore this question in the context of the arguably most fundamental networking task: routing and wavelength assignment.

RWA is conceived and solved in small to medium-sized network topologies using Integer Linear Programming (ILP), which offers efficient (e.g., cost-minimized) solutions to the detriment of complicated, time-consuming, and expensive computations because the problem is understood to be NP-hard. Throughout the literature, suboptimal heuristic algorithms are proposed for speeding up the RWA process throughout large network topologies.

In our study, we divide the routing and wavelength assignment into two problems. In this thesis, we have tried to exploit graph neural networks to solve the routing problem in large optical networks. Graph neural networks are new techniques of machine learning which are recently becoming very popular among researchers in solving network optimization problems. Existing graph Neural Network (GNN) architectures have limited power in capturing the position / location of a given node with respect to all other nodes of the graph while in the problems such as finding the shortest path between a pair of nodes in a graph capturing the position of the nodes is very crucial. In order to address this issue, we have exploited the message passing neural network (MPNN) which is the new class of GNN. In the wavelength assignment, we implemented various heuristic methods and make a comprehensive study. Our results showed that the proposed method can work well for large optical networks and showed reliable results for any kind of network topology and provide the solution in an acceptable time in comparison to heuristics.

Chapter 7

Future Work

Although the provided analyses and methodologies are quite good, there are some improvements that can still be made.

In this context we will survey some of the provided results which can be improved or extended further. This section also briefly describes some interesting research topics, which are worth investigating further. Here are these points:

1. As we have mentioned in the thesis, we divided RWA problem into two separated problems that decreases the complexity of the solution. Our suggested framework to solve the routing problem can also be used for wavelength assignment problem just by adding some other features to the links and nodes. These features can be link betweenness, available capacity in each link and number of connections that are passing across each node. Merging the two steps, can significantly increase the speed of RWA.
2. We have used 10 rounds of message passing in order to get information from others nodes and links in the network in our experiments. Although it has good and satisfying results, but because of large number of computations it increases the computational complexity of the solution. This can be improved if we get the information from randomly selected neighboring nodes and links. It can provide a good overview of the whole network for each node and link. This method called PGNN which is comprehensively described in [17].

Bibliography

[1] Ignacio Martín , Sebastian Troia , José Alberto Hernández , Alberto Rodríguez,Francesco Musumeci , Member, IEEE, Guido Maier, Rodolfo Alvizu , and Óscar González de Dios, "Machine Learning-Based Routing and Wavelength Assignment in Software-Defined Optical Networks," *IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT,* vol. 16, 2019.

[2] Z. M. Fadlullah et al., "“State-of-the-art deep learning: Evolving machine intelligence toward tomorrow’s intelligent network traffic control systems," *IEEE Commun. Surveys Tuts,* vol. 19, p. 2432–2455, 2017.

[3] Jiyang Dong, Junying Zhang, "Neural Network Based Algorithm for Multi-Constrained Shortest Path Problem," *Springer-Verlag Berlin Heidelberg,* p. 776–785, 2007.

[4] Dong, J., Wang, W., Zhang, J., "Accumulative competition neural network for shortest path," *International Conference on Machine Learning and Cybernetics,* vol. III, pp. 1157-1161, 2003.

[5] Woo Young Kwon, Il Hong Suh*, and Sanghoon Lee, "SSPQL: Stochastic Shortest Path-based Q-learning," *International Journal of Control, Automation, and Systems,* pp. 328-338, 2011.

[6] F. M. Tavera, "Dijkstra's Shortest Path Algorithm with step-by-step execution," University of Mexico (UNAM).

[7] Olivier Brun, Sami Baraketi, "Routing and Wavelength Assignment in Optical Networks," *hal,* vol. 01062321, 2014.

[8] C. Natalino, M. R. Raza, P. O¨ hlen, P. Batista2, M. Santos, L. Wosinska, P. Monti, "Machine-Learning-Based Routing of QoS-Constrained Connectivity Services in Optical Transport Networks," *Advanced Photonics Congress,* 2018.

[9] R. I. Muhamedyev, "Machine learning methods: An overview," *COMPUTER MODELLING & NEW TECHNOLOGIES,* pp. 14-29, 2015.

[10] H. M. J. Kairanbay Magzhan, "A Review And Evaluations Of Shortest Path Algorithms," *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 2, ISSUE 6,* June 2013.

[11] A. R. S. Skiena, "Programming Challenges," *The programming Contest training Manual,* pp. 248,250.

[12] C. R. Francesco Musumeci, "An Overview on Application of Machine Learning Techniques in Optical Networks," IEEE, 2018.

[13] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks,* p. 20(1):61–80, 2009b.

[14] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl, "Neural message passing for quantum chemistry," *In International Conference on Machine Learning,* p. 1273–1272, 2017.

[15] Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S. Dux, Ken-ichi Kawarabayashi, "WHAT CAN NEURAL NETWORKS REASON ABOUT?," *ICLR 2020,* 2020.

[16] Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V.F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H.F., Ballard, A.J., Gilmer, J., Dahl, G.E., Vaswani, A., Allen, K.R. et al, "Relational inductive biases, deep learning, and graph networks," *ArXiv,* vol. 3, 2018.

[17] Jiaxuan You,Rex Ying, Jure Leskovec, "Position-aware Graph Neural Networks," in *International Conference on Machine*, Long Beach, California, PMLR 97, 2019.

[18] Palash Goyal and Emilio Ferrara, "Graph Embedding Techniques, Applications, and Performance: A Survey," *arXiv,* vol. 4, p. 1705.02801, December 2017.

[19] Hui Zang,Jason P. Jue,Biswanath Mukherjee, "A Review of Routing and WavelengthAssignment Approaches for Wavelength-Routed Optical WDM Networks," *OPTICAL NETWORKS MAGAZINE,* 2000.

[20] Ryuta Shiraki, Yojiro Mori, Hiroshi Hasegawa, and Ken-ichi Sato, "Dynamic Control of Transparent Optical Networks with Adaptive State-Value Assessment Enabled by Reinforcement Learning," *ICTON,* 2019.

[21] "Github," [Online]. Available: https://github.com/deepmind/graph_nets.

[22] Chien Chen, Subrata Banerjee, "A New Model for Optimal Routing in All–Optical Networks with Scalable Number of Wavelength Converters," *Globecom,* February 1995.