TÉCNICO LISBOA

Designing Ultra-High Bandwidth Optical Networks Using Machine Learning Techniques

Alexandre Paulo da Costa Freitas

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor: Prof. João José de Oliveira Pires

Examination Committee

Chairperson: Prof. José Eduardo Charters Ribeiro da Cunha Sanguino
Supervisor: Prof. João José de Oliveira Pires
Members of the Committee: Prof. Paulo Sérgio De Brito André
Eng. André Nunes de Sousa

June 2024

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First, I would like to thank my dissertation supervisor, Prof. João Pires, for his constant support and guidance during the development of this thesis. His advice and insight were essential in shaping this work.

I would like to thank all my friends who shared this journey at IST with me. Through every challenge and triumph, their presence has been a source of strength and joy, making this academic voyage one to cherish.

Finally, I would like to thank my family for their unwavering support and encouragement. Your belief in me has been a constant source of motivation in my academic pursuit and in all aspects of my life.

Thank you.

Abstract

Optical networks constitute essential infrastructures for telecommunication operators. Given the importance of network capacity in their design, it is paramount to determine it effectively, a process that can involve long computation times. To face this challenge, this thesis proposes a machine learning solution to determine the network capacity of transparent optical backbone networks. This solution is based on a Deep Neural Network (DNN) model, and it aims to provide a fast and reliable way of determining the capacity considering as inputs parameters of the physical topology of the networks. Furthermore, as a way of overcoming the bandwidth limitations of C-band transmission, a solution where optical fibers are added to the network is also implemented. This solution allows for the elimination of the blocking of traffic, effectively leading to the achievement of large network capacities. This second approach led to the development of a second DNN model that predicts the cost in kilometers of fiber deployed, alongside the network capacity. The training of DNN models requires the use of a considerable amount of data, and so a generative graph model that generates networks with a similar topology to optical backbone networks was implemented. A heuristic routing algorithm, also developed in this thesis, is then used to determine the network parameters and capacities. It was shown that the DNN models tend to provide accurate predictions in just a few milliseconds, making the developed models useful tools in the design of optical backbone networks.

Keywords

Transparent optical backbone network, network capacity, random graphs, routing, wavelength and fiber assignment, deep neural networks

Resumo

As redes óticas são infraestruturas essenciais para os operadores de telecomunicações. Dada a importância da capacidade da rede no seu desenho, é fundamental determiná-la de forma eficaz, um processo que pode envolver tempos de computação longos. Para enfrentar este desafio, esta tese propõe uma solução de aprendizagem automática para determinar a capacidade da rede em redes óticas de *backbone* transparentes. Esta solução é baseada num modelo de rede neuronal profunda (DNN) e tem o objetivo de determinar a capacidade de forma rápida e fiável, tendo como inputs parâmetros da topologia física das redes. Além disso, de forma a ultrapassar as limitações de largura de banda da transmissão na banda C, uma solução onde fibras óticas são acrescentadas à rede foi também implementada. Esta solução permite a eliminação do bloqueio de tráfego, levando a capacidades de rede elevadas. Esta segunda abordagem levou ao desenvolvimento de um segundo modelo DNN capaz de prever, para além da capacidade, o custo em quilómetros de fibra. O treino de modelos DNN requer uma quantidade de dados elevada, por isso foi implementado um modelo de geração de grafos que gera redes semelhantes a redes óticas de *backbone*. Um algoritmo de *routing*, desenvolvido também nesta tese, é então usado para obter os parâmetros das redes e as respetivas capacidades. Foi demonstrado que os modelos DNN são capazes de fazer previsões de forma precisa em apenas alguns milissegundos, o que faz destes modelos ferramentas úteis no desenho de redes óticas de *backbone*.

Palavras Chave

Rede ótica de *backbone*, capacidade da rede, grafos aleatórios, *routing*, atribuição de comprimento de onda e fibras, redes neuronais

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

Adam	Adaptive Momentum Estimation
ANN	Artificial Neural Network
ARE	Average Relative Error
ASE	Amplified Spontaneous Emission
BDM	Band Division Multiplexing
BER	Bit Error Rate
CWDM	Coarse Wavelength Division Multiplexing
DNN	Deep Neural Network
DWDM	Dense Wavelength Division Multiplexing
EDFA	Erbium-Doped Fiber Amplifier
ILP	Integer Linear Programming
ML	Machine Learning
MSE	Mean Squared Error
MST	Minimum Spanning Tree
NLI	Non-linear Interference
OOD	Out-of-Distribution
OSNR	Optical Signal to Noise Ratio
PM	Polarization Multiplexed
QoT	Quality of Transmission
ReLU	Rectified Linear Unit
RMSA	Routing, Modulation format, and Spectrum Allocation
ROADM	Reconfigurable Optical Add-Drop Multiplexer

RWA	Routing and Wavelength Assignment
RWC	Routing and Wavelength Configuration
SDM	Space Division Multiplexing
SGD	Stochastic Gradient Descent
SHAP	SHapley Additive exPlanations
SNR	Signal-to-Noise Ratio
WDM	Wavelength Division Multiplexing

Chapter 1

Introduction

1.1 Motivation

Optical networks are telecommunication infrastructures in which data is transmitted using light, typically in the infrared region of the spectrum, through optical fiber links. Since optical networks offer very high bandwidths and allow for transmission over great distances, they have become increasingly more ubiquitous over the last decades due to the rapid pace at which telecommunications traffic has been growing [1]. Predictions indicate that this trend is expected to continue for the next years as applications and services that generate large amounts of data become more widespread. This makes optical networks indispensable in the evolving telecommunications landscape, making the development of solutions related to this type of networks particularly relevant.

Optical networks can be divided into three different tiers depending on their geographic extension and capacity [2]: (a) backbone networks, which cover the largest distances, spanning hundreds to thousands of kilometers, and carry traffic from millions of users, offering very high capacities (in the order of dozens of Tbit/s); (b) metro networks, which cover cities and metropolitan areas, spanning tens to hundreds of kilometers, and offer capacities in the order of the hundreds of Gbit/s; and (c) access networks, which encompass smaller areas (few kilometers), connecting end-users to the network providers and delivering data rates on the order of a few Gbit/s.

Optical networks can also be classified as opaque, transparent or translucent. In opaque networks, the node functions (such as multiplexing, switching, routing, etc.) take place in the electrical domain, while in transparent networks this is done in the optical domain. In translucent networks there are nodes operating in the electrical domain and others operating in the optical domain.

The work developed in this dissertation project focuses on transparent optical backbone networks.

A fundamental technology in the field of optical networking is Wavelength Division Multiplexing (WDM). WDM enables the simultaneous transmission of multiple optical signals (also designated as

optical channels) on the same optical fiber, with each channel using a different wavelength. The number of optical channels being transmitted in a fiber depends on the spacing between the channels as well as the bandwidth of the WDM signal. Optical backbone networks typically operate in the C-band, which has a bandwidth of approximately 4800 GHz. This bandwidth corresponds to the bandwidth of the Erbium-Doped Fiber Amplifier (EDFA), which is the most commonly used optical amplifier, being used to compensate for the fiber losses [3].

In order to significantly increase the number of optical channels in a network, one can utilize Band Division Multiplexing (BDM) or Space Division Multiplexing (SDM). The first technique explores the use of bands other than C-band, while the second one relies, for example, on adding more optical fibers to the network's links, while still operating in the C-band.

The capacity of an optical channel is the maximum amount of data per unit time at which the information can be transmitted through a noisy medium without errors. The capacity of an optical network is the maximum data rate of the entire network, that is, it is the sum of the capacity of all of the optical channels. These two concepts are crucial when it comes to the analysis and design of optical networks, as the capacity can be an essential way of assessing a network's performance, allowing for the optimization of network parameters and resources.

The determination of the optical channel capacity is, however, a somewhat complex problem, due to the presence of non-linear effects in optical fiber transmission. There are mathematical and analytical models which aim to accurately estimate the channel capacity, but these can be complex, often involving long computation times. Besides the physical layer characteristics, the determination of the capacity of an optical network also needs to take into account aspects related to the network layer, such as the network topology, the traffic demands and routing, making it an even more complex procedure.

The problem of evaluating the capacity of optical networks has received some attention recently (see [4] and references therein) using either Integer Linear Programming (ILP) or appropriate heuristics. But even when using heuristics the computation time required to calculate the capacity of large networks remains very high.

To address these limitations, it is worthwhile to consider using machine learning techniques, such as Deep Neural Networks (DNNs) for computing the capacity of optical networks. Although machine learning techniques have been used widely in the field of optical networks [5, 6], to the best of the author's knowledge, there are no reported applications in the context of network capacity evaluation.

1.2 Objectives

The main objective of this dissertation is to develop a machine learning model, a DNN, able to determine the network capacity and the average channel capacity of transparent optical backbone networks operating in the C-band, accounting for the aspects related to both the physical and the network layer.

Specifically, this model should be able to predict these outputs based on input parameters related to the network's physical topology in a fast and accurate manner.

Training a DNN requires a large amount of data. For that reason, the use of a generative graph model is essential, as it provides a way of obtaining thousands of artificial networks from which that data can be obtained. Therefore, another important objective of this dissertation project is the implementation of a generative model capable of generating graphs whose topology is similar to that of optical backbone networks.

The DNN's training dataset is composed of features and labels. The features are the inputs of the model, the parameters related to the physical topology of the network, and the labels are the outputs of the model, that is, the total network capacity and the average channel capacity. The determination of features and labels is a process that involves the consideration of the physical layer aspects (which were accounted for through the optical reach), and the network layer aspects, for which a routing algorithm needs to be used. For that reason, the development of a program capable of determining the various network parameters as well as find the average channel capacity and network capacity through the implementation of a routing algorithm is also one of the main goals of this dissertation project.

Although one of the initial objectives of this thesis was to explore the use of BDM (multi-band transmission on the S+C+L bands) as a way to overcome the saturation of the C-band and achieve ultra-high bandwidth transmission, the complexity of the physical modeling of these networks motivated the author to instead explore the use of SDM to overcome that limitation.

Consequently, two types of routing algorithms were developed: (a) a constrained routing algorithm that takes into account the limitations in the number of optical channels in the optical fibers; and (b) an unconstrained routing algorithm with fiber assignment, a routing solution where the limitation in the number of optical channels was overcome through the introduction of additional optical fibers to the network's links. From this second case, it was possible to develop a DNN model where the outputs are the (unconstrained) network capacity and the total cost in kilometers of fiber deployed.

1.3 Tools

The development of the models, algorithms and auxiliary code was done in *Python*, as this language provides various libraries that offer very useful features when it comes to network analysis, implementation of DNNs, as well as general mathematical and graph plotting tools. The main libraries used were: (a) *NetworkX*, a package for network analysis and exploration which provides data structures to represent many types of networks, many tools and algorithms for calculating network properties, as well as the drawing and visualization of the graphs [7]; (b) *Pytorch*, a library for machine learning applications that makes use of tensor computation and allows for an efficient implementation and training of DNNs [8]; (c) *Numpy*, a library for mathematical operations on multi-dimensional arrays and matrices;

(d) *MatPlotlib*, a plotting library used for the creation of static and dynamic visualizations; and (e) *shap*, a library used to analyze the importance of the features (inputs) on the predictions of a DNN.

1.4 Contributions

The work developed in this thesis resulted in the development of DNN models whose main goal is to assist in the analysis and design of optical networks, allowing for the determination of the network's capacity in a fast and reliable way. Furthermore, as a result of the development of the DNN models, other software tools were also developed. These tools also proved to be useful in the analysis of optical backbone networks.

A generative graph model that is capable of generating graphs with topologies similar to those of optical backbone networks was implemented. This model was based on the model described in [9], but some changes were made to assure that the generated topologies more closely resemble their real-world counterparts, to improve the overall functioning of the model given the specificities of this work, and also when the described implementation was lacking in detail. The developed model is able to generate a large number of artificial random networks in short computation times, making this a useful tool for applications that require the use of a large amount of network topologies.

A program to determine various parameters of a network, as well as the total network capacity, average channel capacity, and the cost in kilometers of fiber, was developed as a way of obtaining the data to train the DNN models. This program implements the routing algorithms, allowing for the specification of the maximum number of optical channels per fiber, as well as the possibility of adding optical fibers to the network (SDM solution) as a way to overcome this limitation. For the SDM solution, two fiber assignment algorithms were developed, one focused on computational performance (which is essential when determining the parameters of thousands of networks), and the other where a more comprehensive analysis on the placement of fibers is made, leading to less fibers being introduced in the networks (at the expense of longer computation times). The development of this routing program has resulted in a comprehensive tool for network analysis that serves as a platform for facilitating a deeper understanding of optical networking.

Through the use of both of these programs, it was possible to produce an article focused on the topic of capacity in optical backbone networks, analyzing how the different network and physical layer parameters influence this value. This article was published on the journal "Photonics":

> A. Freitas and J. Pires, "Investigating the Impact of Topology and Physical Impairments on the Capacity of an Optical Backbone Network," Photonics, vol. 11, no. 4, 2024. [4]

1.5 Organization of this Document

The remainder of this thesis is organized as follows:

- **Chapter 2 (Fundamental Concepts and State of the Art):** This chapter lays the theoretical groundwork that serves as a basis for the work presented in this thesis. It delves into network characterization, discussing various network parameters relevant to this work, as well as specific concepts of optical networks and the determination of network and channel capacity. It also explores the theoretical aspects of DNN models and the state of the art of machine learning applications in the context of optical networking, with a particular focus on DNN models.
- **Chapter 3 (Generation of Random Networks):** In this chapter the concept of random network is explained and the generative model implemented in this thesis is detailed, the key attributes of the generated random networks are compared to their real-world equivalents.
- **Chapter 4 (Routing, Wavelength Assignment and Fiber Assignment):** This chapter provides an explanation of the routing algorithms employed in this thesis, including the scenario where the SDM solution is considered. It presents and discusses the results of their application on both reference and artificial random networks, including the analysis of the routing solutions in relation to different network parameters.
- **Chapter 5 (Implementation of the DNN Models):** This chapter describes the process of training the two DNN models and presents the tests performed to both models on sets of random networks as well as reference networks. It also includes an analysis of the importance of the DNN model's features.
- **Chapter 6 (Conclusion):** The final chapter summarizes the conclusions drawn from the work conducted in this thesis and discusses potential areas for future development.

Chapter 2

Fundamental Concepts and State of the Art

This chapter aims to provide a comprehensive overview of the essential principles that underpin the work developed in this thesis, as well as review the current state of the art of machine learning applications in the context of optical networking, highlighting recent advancements. In this chapter, topics related to the representation of networks, routing, the characteristics of optical networks, determination of network and channel capacity, and the theoretical aspects of neural networks are addressed.

2.1 Network Characterization

A network can be defined, in a general way, as a group of elements that are interconnected or interrelated. Networks can be represented by graphs, which are sets of connected elements called nodes (or vertices) that are usually represented by points. The connections between the nodes are called edges (or links) and they are usually represented by lines that connect the nodes.

In telecommunication networks, the network nodes correspond to electronic or optical devices (such as computers, routers, switches, multiplexers, etc.) that are responsible for sending, receiving or routing the information. The network links, used to connect the nodes, can be based on cable media (twisted pair, coaxial cable, optical fiber, etc.) or can be wireless (radio waves) [10]. In transparent optical backbone networks, the networks considered in this thesis, the links are implemented with optical fiber cables and the nodes with Reconfigurable Optical Add-Drop Multiplexers (ROADMs), which are optical devices that allow for the switching of traffic at the wavelength level without opto-electric conversion.

2.1.1 Representation of Telecommunication Networks

A telecommunication network can be represented by its physical topology and its logical topology.

Physical Topology

The physical topology of a network describes the physical layout of the various elements of the network and it can be represented as a graph, $G = (V, E)$, with $V = \{v_1, ..., v_N\}$ being the set of vertices and $E = \{e_1, ..., e_K\}$ the set of edges, where N is the number of vertices and K is the number of edges.

In a network, the links can have attributes associated to them, such as length, capacity, cost and others. When a graph has links with such attributes (also called weights) it is called a weighted graph, otherwise it is called an unweighted graph. Figure 2.1 represents a weighted graph where the weights are the lengths (in kilometers) between the nodes.

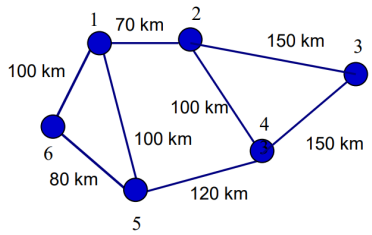


Figure 2.1: Example of a Weighted Graph. (From [2])

Graphs can be directed or undirected depending on whether their links are unidirectional or bidirectional. A unidirectional link is one in which communications are only done one-way, so the order of the nodes in the link matters (for example, communication is done from node v_1 to v_2, but not the other way around). Unidirectional links are also called arcs. In a bidirectional link, the communications are done in both ways. Consequently, a bidirectional link can be seen as a pair of arcs with opposite directions but equal weights. In the case of optical networks, the bidirectional links have a given number of fibers in one communication direction and (typically) the same number in the opposite direction.

In the work developed on this thesis, the graphs considered are weighted and undirected graphs where each weight represents the physical distance between the nodes, being designated as link length. Accordingly, one can also define the concept of average link length, which is calculated as the mean length across all the links in a network.

The physical topology of a network can also be described through an adjacency matrix. The adjacency matrix of a network with N nodes, A, is a square $(N \times N)$ matrix that represents a graph. For unweighted graphs, the element a_{ij} of the matrix is 1 if there is a link between the nodes v_i and v_j. For weighted graphs, the element a_{ij} will have the value of the weight itself. In either case, if there is no link between the nodes, the value of the element a_{ij} is zero. Since links from a node to itself are not allowed

in simple graphs, the diagonal elements of the adjacency matrix (the elements a_{ij} with $i = j$) will also be zero. If a graph is undirected, its adjacency matrix will be symmetric. Nodes are considered adjacent if there is a link connecting them directly. Links are adjacent if they share a common node.

Logical Topology

The logical topology of a network describes how the transmission of data (the flow of traffic) occurs. This flow is described by the number of traffic demands (or logical links). The logical links can be unidirectional or bidirectional and can exist between adjacent or non adjacent nodes. Figure 2.2 represents the physical and logical topologies of a network, with the logical links marked in red.

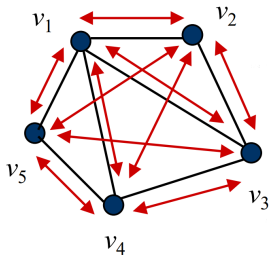


Figure 2.2: Example of Logical Topology. (From [10])

Another way of representing a logical topology of a network is with a demand matrix. A demand matrix, D, is a square $(N \times N)$ matrix where the element d_{ij} is equal to 1 if there is a traffic demand between nodes v_i and v_j, or zero if that demand doesn't exist. When there is a demand from each node to every other node, it is said to be a logical full-mesh topology, and there are $N(N-1)$ unidirectional traffic demands.

Related to the logical topology is the concept of network traffic. Network traffic represents the amount of data flowing through a network in a given instant or period of time. The traffic can be classified as static (when the traffic demands are known *a priori* and remain constant in time) or dynamic (when traffic demands arrive randomly and are released after some time, being that the instantaneous traffic between a pair of nodes randomly fluctuates around an average value).

In the case of static traffic, the case considered in this work, the traffic demands can be described through a traffic matrix. A traffic matrix, T, is a square $(N \times N)$ matrix where the element t_{sd} represents the traffic intensity from node s (source) to node d (destination). The traffic intensity is the volume of traffic over a period of time (in, for example, bit/s or packets/s).

2.1.2 Network Parameters

Having in consideration both the physical and logical topology, there are several parameters that are relevant when analyzing a network. The definition of these parameters is essential in the context of this

thesis, as they are utilized throughout various parts of this work, including the routing, the generation of random networks, and the implementation of the DNN models.

In a graph, a path π_{sd} is a sequence of adjacent links and nodes that begins in a source s and ends in a destination d. It can be described by either the sequence of its nodes or links, and it can be directed or undirected depending on whether its components are arcs or bidirectional links. In the context of optical networks, a path represents the physical route of an optical channel, defining the sequence of links/nodes from s to d.

The length of a path, $l(\pi_{s,d})$, in a weighted graph is the sum of the lengths of all the links in that path. If the graph is unweighted the weight of each link is one and so the length of the path is the number of links in the path. The distance between two nodes is the length of the shortest path between them. This concept is very important when it comes to routing, as routing is usually done across the shortest paths. One of the most relevant algorithms to compute shortest paths is *Dijkstra*'s algorithm (that finds the distance from a source node to all other nodes).

The diameter of a network is the maximum number of links between any pair of nodes considering shortest-path routing, i.e., the number of links in the longest shortest path.

The node degree, δ_i, is the number of links incident on node v_i, and the average node degree of a network, $\langle\delta\rangle$, can be obtained by computing the average of the node degree across all the nodes in the network according to the following expression:

$$\langle\delta\rangle = \frac{1}{N}\sum_{i=1}^{N}\delta_i \ . \tag{2.1}$$

The number of hops per demand is the number of links that are traversed by a demand (as established according to a given routing strategy). Usually, the number of hops per demand is defined by the shortest path between the source and destination nodes of a given traffic demand. The hop matrix H is a square ($N \times N$) matrix where the element h_{ij} is the number of hops traversed by a traffic demand between nodes v_i and v_j. The average number of hops per demand can be computed from the hop matrix through the following expression (where n_D is the number of bidirectional demands):

$$\langle h\rangle = \frac{1}{n_D}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N} h_{ij} \ . \tag{2.2}$$

A relevant statistical measurement that is used as a way to analyze network parameters is the variance. The variance is used to indicate the dispersion of data points in a dataset around the mean. It quantifies the degree of variation or diversity present in a given network parameter. It is given by:

$$\mathrm{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)^2 \ . \tag{2.3}$$

where x_i represents the parameter value, μ is the mean of the parameter values and n is the total number of observations. The variance of the link length and the variance of the node degree were two parameters considered in this thesis.

The traffic load in a given link is the average value of traffic flowing through that link over a certain period of time. In static traffic situations, the value of the load will simply be the sum of all the traffic in that link (since the traffic does not vary over time). The average load per link is the average of the traffic loads in every link of the network. These parameters can only be computed after having previously defined the paths for all traffic demands in the network through some routing strategy.

The capacity of a link is the maximum amount of traffic that link can transport. It is related to the channel capacity (C_{ch}), as the channel capacity is limited by the smallest link capacity in the path of that optical channel (as this link's capacity will impose a limit on the maximum traffic of the channel).

The connectivity of a network is a way of measuring the network's fault tolerance or resilience. The node connectivity (κ-connectivity), $\kappa(G)$, is the minimum number of nodes that have to be removed from a graph to disconnect it, that is, separate it into two or more subgraphs with no links between them (resulting in at least two nodes that can't be connected through a path). Similarly, the edge connectivity (λ-connectivity), $\lambda(G)$, is the minimum number of edges that have to be removed from a graph to disconnect it.

The algebraic connectivity is the second smallest eigenvalue of the graphs's Laplacian matrix. The Laplacian matrix is defined as $L = D_\delta - A$, where A is the graph's adjacency matrix and D_δ is the degree matrix (a diagonal matrix that contains the node degree for each node) [11, 12]. This parameter is related to the robustness of a network: larger algebraic connectivity values are associated to more robust networks, meaning that these networks are better connected and thus removing a node or a link is less likely to disconnect the network [11].

2.1.3 Routing

Routing is the process which is responsible for mapping the logical topology on the physical topology; that is, it's the selection of the path ($\pi_{s,d}$) to be followed by a traffic flow associated with a specific traffic demand. There are two types of routing problems: the capacitated routing problem (also referred to as constrained routing) and the uncapacitated routing problem (or unconstrained routing).

In the capacitated routing problem, the objective is to maximize the allocated traffic for a given physical topology, a given link capacity, and a given traffic matrix, with the requirement that the traffic in each link cannot exceed its capacity. During the capacitated routing, if there are no links with enough residual capacity (the difference between the link capacity and its load) for a given traffic demand to be routed, then that traffic demand is blocked. In this case, the blocking probability can be determined by dividing the number of blocked traffic demands by the total number of traffic demands.

In the uncapacitated routing problem, the objective is to find, for a given physical topology and traffic matrix, the set of paths that support all the traffic demands and to determine the capacities needed for each of the links in the network.

Given that there are different possible paths from a source to a destination, most routing strategies involve some sort of shortest path algorithm, trying to minimize a particular metric (like number of hops, total path length, etc.).

In either case, there is a general strategy for routing that can be followed for static traffic problems. Given as input parameters the physical topology described as a graph $G(V, E)$ and a traffic matrix T, one should perform the following steps to route the traffic through the network:

- Find the shortest paths for all traffic demands using a heuristic algorithm (like *Dijkstra*'s).
- Order the demands according to a sorting strategy.
- Route the demands according to the previous ordering. To break a tie, choose the path that minimizes the load in the most loaded link.

To order the traffic demands there are the following strategies:

- Shortest-First: The demands with the lowest number of hops/shortest path length come first;
- Longest-First: The demands with the highest number of hops/longest path length come first;
- Largest-First: The demands with the highest number of traffic units come first.

The specific implementations of the routing algorithms developed in this dissertation project are detailed in Chapter 4.

2.2 Optical Networks

2.2.1 Optical Bands and WDM

Optical networks are telecommunication networks in which the information is transmitted using electromagnetic signals through optical fiber links. These networks operate in the electromagnetic spectrum mainly in the optical bands between 800 nm and 1600 nm. This choice of bands ensures that the attenuation the signal suffers while traveling through the fiber is minimized, as defined by the attenuation coefficient. The attenuation coefficient, α (in dB/km), is the rate at which the signal's power decreases per unit distance, being caused by factors such as absorption, scattering, and bending losses. Figure 2.3 shows the attenuation coefficient in relation to the wavelength usually observed in legacy optical fibers (like the ITU-T G.652.A) as well as the different optical bands.

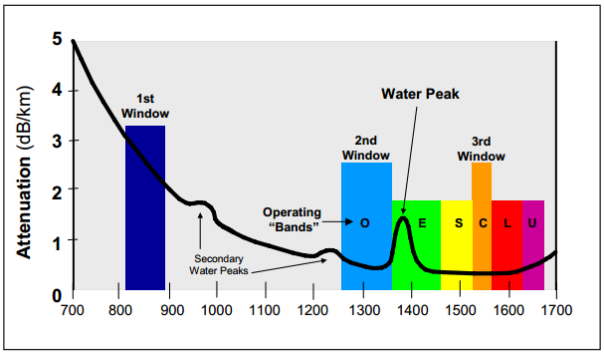


Figure 2.3: Typical Fiber Attenuation (dB/km) relative to the wavelength (nm). (From [13])

The first transmission window (in the 800 to 900 nm range) is typically used in multi-mode fibers, designed for short distance. The main benefit of using such systems is their lower cost. Single-mode optical fibers, nowadays more commonly used in long distance telecommunications, operate in the O, E, S, C and L bands.

More specifically, backbone networks, which cover the greatest distances, typically operate in the C-band, the band where attenuation is lowest, and for which amplification technology is more mature, with EDFAs being commercially available for some decades (although L-band EDFAs have also been available for some time).

As previously mentioned, optical backbone networks make use of WDM so that different signals (optical channels) can share the same optical fiber, each signal operating with a different wavelength. There are two types of WDM: Coarse Wavelength Division Multiplexing (CWDM) and Dense Wavelength Division Multiplexing (DWDM). CWDM systems are WDM systems with few active wavelengths per fiber. It employs wide-range channels with wavelengths that are spread apart, allowing for wavelength drift. In DWDM systems, there are more active wavelengths per fiber as the spacing between channels is smaller. While these systems are more expensive to implement and operate, a more efficient use of the spectrum is made, being possible to fit dozens of wavelengths of fixed spacing into the C-band [14, 15]. Figure 2.4 shows a representation of WDM.

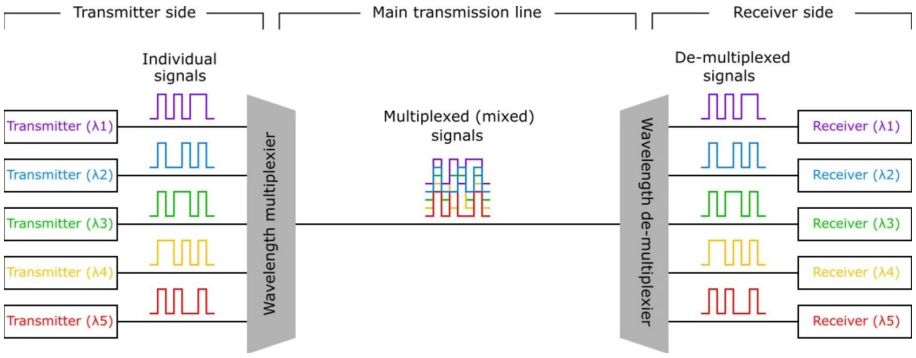


Figure 2.4: Wavelength Division Multiplexing (From [15])

Each individual signal represented in Fig. 2.4 represents an optical channel. An optical channel can be defined as the communication pathway through which data is transmitted, in the optical domain, from a sender (s) to a destination (d), being characterized by its carrier wavelength λ_c (or carrier frequency ν_c), bandwidth B_{ch} (in Hz), and bit rate R_b (in bit/s). Note that the bit rate can be related to the symbol rate R_s (in baud) through $R_b = R_s \cdot \log_2(M)$, where M is the number of symbols of the modulation scheme used. Figure 2.5 shows a representation of the optical channels in a WDM system.

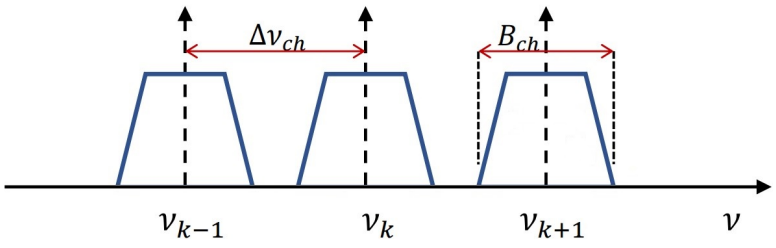


Figure 2.5: Optical channels in a WDM system (in the frequency domain).

In Fig. 2.5 there are three optical channels, whose carriers have the frequencies ν_{k-1}, ν_k and ν_{k+1}. The bandwidth of the channel (B_{ch}) as well as the channel spacing ($\Delta\nu_{ch}$) are also represented. Considering transmission in the C-band (with bandwidth of approximately 4800 GHz), for a typical channel spacing of 50 GHz, an optical fiber can support up to 96 channels [3]. The minimum bandwidth that guarantees a signal transmission over the fiber without inter-symbol interference is equal to the symbol rate R_s, as defined by the Nyquist criterion [16]. So, in that case, for transmission at 64 Gbaud (a typical value), the bandwidth B_{ch} should be 64 GHz, as well as the channel spacing $\Delta\nu_{ch}$. This allows for the co-existence of 75 optical channels in the same optical fiber.

Besides the transmission in the C-band, it is possible to use other optical bands in the low-loss optical spectrum of single mode fiber, allowing for a much wider bandwidth and, as a consequence, for the ability to transmit more optical channels per optical fiber. This is called multi-band optical transmission, being based on BDM. It was shown in [17] that multi-band transmission in the S+C+L-bands allowed for a five times increase in the number of optical channels over C-band only transmission, going from 80 optical channels in the C-band to 400 in the S+C+L-bands. This resulted in an increase of the network capacity of $\sim 320\%$ [17].

Currently, transmission in the extended C+L-band is already being used in submarine cable applications. The move towards other optical bands will be done gradually, as the need for more bandwidth appears, with the next stage likely being transmission in the S+C+L-bands. This type of transmission still has, however, its technological challenges, with different network components and technologies that need to be adapted for multi-band, such as, coherent transceiver front-ends, optical amplifiers and wavelength selective switches [18].

2.2.2 Capacity in Optical Networks and Optical Reach

In the context of this dissertation project, two of the most fundamental concepts are the optical channel capacity and the network capacity.

The capacity of an optical channel (C_{ch}) is the maximum data rate at which information can be transmitted through the physical medium (optical fiber), being usually expressed in bit/s. The network-wide average channel capacity is defined in [19] by the following expression:

$$\bar{C}_{ch} = \frac{\sum_k C_{ch,k}}{\sum_k \gamma_k}, \tag{2.4}$$

where γ_k is the expected utilization ratio of a channel k. Usually, for simplicity, γ_k is considered to be 1, and so, the sum in the numerator of Eq. (2.4) is equal to the total capacity of the network and the sum in the denominator is the total number of channels in the network (excluding the optical channels associated to blocked traffic demands).

The capacity of a channel (C_{ch}) can be determined through the Shannon capacity theorem [20], under the assumption that the noise sources are modeled as additive, white and gaussian noise sources and considering the channel bandwidth (B_{ch}) equal to the symbol rate (R_s), being given by [21]:

$$C_{ch} = 2 \cdot R_s \cdot \log_2(1 + \mathit{SNR}), \tag{2.5}$$

where we have the Signal-to-Noise Ratio (SNR) at the receiver end, given by [21]:

$$\text{SNR} = \frac{P_{ch}}{N_0 \cdot R_s}, \tag{2.6}$$

with P_{ch} being the received average optical power per channel (in W), and N_0 the noise power spectral density (in W/Hz). In Eq. (2.5) the factor 2 results from the fact that in optical transmission it is possible for two optical channels to be transmitted in the same fiber, at the same frequency, by using two orthogonal polarization states, being referred to as Polarization Multiplexed (PM) optical channels.

It is through the SNR that various physical aspects of the transmission of an optical channel are taken into account and so, the determination of the SNR involves the consideration of the Amplified Spontaneous Emission (ASE) noise (N_{ASE}), which is a result of optical amplification, as well as the noise resultant from Non-linear Interference (NLI) (N_{NLI}). The work developed in [4] presents a characterization of the N_{ASE} and N_{NLI}, and reaches expressions to determine the SNR and the optical channel capacity that depend on various parameters related to the physical aspects of optical transmission.

In this thesis, as a way to simplify the determination of the channel capacity and reduce computation time, as well as streamline the analysis of the results, the computation of the optical channel capacity will be done through the optical reach.

The optical reach (or transmission reach) can be defined as the maximum transmission distance over which the optical channel can maintain a data rate close to its Shannon capacity. In the context of this work, it is used to account for the impact of the physical layer on the performance of the optical channels. Table 2.1, from [4], shows the optical reach for different values of Shannon channel capacities (referred to as such because they are obtained using the Shannon theory) for transmission at 64 Gbaud and 128 Gbaud.

Table 2.1: Optical Reach and Shannon Capacity. (From [4])

64 Gbaud		128 Gbaud	
Reach (km)	**Capacity (Gbit/s)**	**Reach (km)**	**Capacity (Gbit/s)**
23120	200	20808	400
11120	300	10008	600
5840	400	5256	800
3280	500	2952	1000
1760	600	1584	1200
1040	700	936	1400
560	800	504	1600
320	900	288	1800
160	1000	144	2000
80	1100	72	2200

Knowing the length of the optical channels (which are determined in the routing process), it is possible to approximate the capacity of the optical channel through Table 2.1. For example, for transmission at 64 Gbaud, if the length of a given optical channel $l(\pi_k)$ is 400 km, then the largest capacity value in Table 2.1 that is able to accommodate the 400 km requirement is chosen. In this case, that capacity corresponds to 800 Gbit/s (and it allows for transmission up to 560 km).

Having the capacity of all the optical channels allows for the determination of the average channel capacity $\bar{C}_{ch}$ with Eq. (2.4). The total network capacity can then be determined with:

$$C_{net} = \bar{C}_{ch} \times (N_{ch} - N_{\text{blocked}}), \tag{2.7}$$

where N_{ch} is the total number of traffic demands and N_{blocked} the number of blocked traffic demands (in the cases of constrained routing). In the case of a full-mesh logical topology, there will be $N(N-1)$ unidirectional optical channels. In that case we have:

$$C_{net} = \bar{C}_{ch} \times N(N-1) \times (1 - \bar{B}), \tag{2.8}$$

where $\bar{B}$ is the average blocking ratio, given by:

$$\bar{B} = \frac{N_{\text{blocked}}}{N(N-1)} \, . \tag{2.9}$$

2.2.3 Quality of Transmission

In optical communications the optical signals are susceptible to degradation resultant from physical effects of the transmission. These phenomenons result in the decrease of the signal quality, which can be assessed through different Quality of Transmission (QoT) metrics. These metrics allow for a rigorous quantification of the signal quality, being useful to evaluate the performance of the optical communication system, optimize its design, and ensure reliable data transmission over long distances. The Shannon Capacity and the SNR, already described in previous sections, are QoT metrics used in this work, but other QoT metrics that are also widely used include, for example, the Bit Error Rate (BER) and the Q-factor.

The BER is the ratio between the number of bits received in error and the total number of bits received, in a given period. It can be related to the Q-factor by the following expression [22]:

$$BER = \frac{1}{2} erfc\left(\frac{Q}{\sqrt{2}}\right), \tag{2.10}$$

where Q is the Q-factor and *erfc* is the complementary error function. Assuming that the bandwidth of the electrical receiver is equal to the symbol rate, the Q-factor in dB is given as:

$$Q[dB] = 20\log(\sqrt{\text{SNR}}), \tag{2.11}$$

where SNR is given by Eq. (2.6).

2.3 Machine Learning Applications in Optical Networks

Machine Learning (ML) is the a field of study that deals with the development and study of algorithms that can learn from data and generalize to unseen data, being able to accomplish tasks without direct instructions. ML algorithms can be grouped into four main categories: supervised learning, unsupervised learning, reinforcement learning and semi-supervised learning. While all of these types of ML methods have their applications in the context of optical networks, supervised methods are the most common [5].

In supervised learning, the goal is to predict the value of one or more output variables, given a set of input values (typically a vector of input variables x). When the output variables are continuous, that ML problem is called a regression problem. When the output variables are discreet, it is a classification problem. In this thesis, since the values being predicted (average channel capacity, total network capacity and total fiber cost) are continuous, the problem at hand is a regression problem.

Neural networks, also called Artificial Neural Networks (ANNs), are a particularly relevant example of a supervised learning model. In ANN models, functional transformations are applied to the set of inputs with the goal of predicting outputs or discovering patterns in the data. These networks are constituted

of neurons (or units) that apply a non-linear function to the combination of the inputs and are separated into different types of layers: input, hidden, and output layers. Deep Neural Networks (DNNs) are neural networks with multiple hidden layers. These additional layers enable DNNs to model more complex, non-linear relationships in the data, enhancing their predictive capabilities. In fact, it was proved that DNNs with at least one hidden layer can be used to approximate any continuous function arbitrarily well, as long as there is a sufficient number of neurons [23].

Given their universal approximation characteristic, along with their proven efficacy in numerous optical networking problems (detailed in Section 2.3.1), offering high accuracy and short prediction times, the DNN algorithms stand out among other supervised learning algorithms. Therefore, for the purposes of this thesis, DNNs were the preferred choice.

2.3.1 State of the Art of Machine Learning Applications in Optical Networks

The applications of ML techniques in the context of optical networking are very extensive. In [5] an overview of this type of applications is presented. According to this analysis, the growing trend of applying ML techniques in the context of optical communications and networking are explained by two factors: increased system complexity and increased data availability.

The increased system complexity is the result of the adoption of advanced transmission techniques with a large number of adjustable parameters (such as modulation formats, symbol rates, adaptive coding rates, adaptive channel bandwidth, etc.), which makes the modelling of the system through closed-form formulas very difficult. Analytical models that introduce error margins are commonly adopted in these cases, but this leads to resource under-utilization. ML methods are a good solution for this problem, as they can capture complex non-linear system behavior by exploiting the network data. The increased availability of data results from the prevalent data monitoring in modern optical networks. Data such as traffic patterns, signal quality indicators, failure detection, user's behavior, etc., provide a large amount of data that can be used to train ML algorithms.

In [5] various ML applications are looked at, according to two domains: the physical layer domain and the network layer domain.

When considering the physical layer applications, ML is used mainly due to the presence of non-linear effects in the optical fiber transmission, that make the analytical models either too complex (and, as a consequence, having high computation times) or too inaccurate. In this context, ML is mainly used to estimate the QoT (through metrics such as the SNR, BER and Q-factor), but other applications include: control of optical amplifiers, modulation format recognition, non-linearity mitigation or optical performance monitoring [5].

In the network layer domain, ML is mainly used in network design and management. Given the growing increase of both traffic and the traffic requirements (in terms of, for example, capacity, latency or

quality of service), optimization techniques and traffic engineering methods have become increasingly complex, often involving high computational times. ML techniques allow for the automation of network configuration, ideal for real-time network adjustments. Use cases related to the network layer include: traffic prediction, virtual topology design and reconfiguration, failure management, traffic flow classification or path computation [5].

In [24], an analysis is made on various ML applications for the estimation of QoT, with a strong focus on neural networks, highlighting the low prediction errors and short computation times these models offer in both classification and regression problems. The applications presented in this article include the estimation of BER, SNR or Q-factor based on transmission line parameters (transmission power, number of spans, inter-channel spacing, etc.) or network related parameters (number of links, average link length, number of nodes, etc.).

Some relevant examples of regression models presented in [24] include: (a) an ANN model used to estimate the SNR based on the inputs channel power and frequency, number of spans and ASE and nonlinear noises, which resulted in predictions with a maximum error < 0.5 dB; (b) a DNN model used to predict the Q-factor based on launch power, laser bias and EDFA input/output powers, which allowed for estimations with root mean squared errors < 0.02 dB; and (c) an ANN model to estimate the SNR with a set of inputs that includes parameters such as fiber attenuation, dispersion coefficient, span length, number of active channels, launch power, channel bandwidth and frequency, etc., which resulted in over 99.9% of the estimates having an error within 0.5 dB, with computation times < 10 ms .

In [25], various ML models are used to estimate the QoT, specifically the SNR and Optical Signal to Noise Ratio (OSNR) in the context of WDM optical transmission (being regression problems). The ML techniques considered were: polynomial regression, random forest regression, and ANNs. The outputs were treated separately, so different models of each ML technique were developed to predict each output. The input parameters of all these supervised methods were 11 features related to transmission and network characteristics: transmission power at the beginning of the optical line (dBm), total chromatic dispersion (ps/nm.km), total polarization mode dispersion (ps), total polarization dependent loss (dB), number of channels, number of fiber spans, longest link length (km), total network length (km), symbol rate (Gbaud), bitrate (Gb/s), and OSNR at transceiver output (dB).

It was concluded that the ANN models outperformed the other ML techniques in the estimation of both outputs, offering a good compromise between model complexity and performance. The ANN models were able to achieve these results even with a single hidden layer and a relatively low number of neurons (7 and 5 neurons for the SNR and the OSNR prediction models, respectively).

In [26], the effectiveness of four types of ML models is evaluated in the context of predicting the QoT of optical channels (referred to as lightpaths) in DWDM networks working on the C-band. The QoT is assessed through the residual margin, defined here as the difference between the OSNR of a lightpath

at the end of the path and the required OSNR in back-to-back operation to achieve acceptable system performance (with the OSNR being determined, in this context, according to Eq. (16) and Eq. (17) of [26]). If the residual margin is ≥ 0, it means that the lightpath has a higher OSNR than required (or just what is required, if equal to 0), indicating that the lightpath is feasible and can support data transmission with acceptable performance. On the other hand, if the residual margin is negative, it means that the lightpath's OSNR is lower than what is required for acceptable performance. This indicates that the lightpath is not feasible as it may not support data transmission with acceptable performance.

The models considered were: K-nearest neighbors, logistic regression, support vector machines, and two ANN models. These models were used to solve a classification problem where the output was 1 if the residual margin is ≥ 0, and 0 otherwise. But one of the ANN models was a regression model that predicts the residual margin value in dB. In both ANN models the neural networks were comprised of one hidden layer with 12 neurons. All the ML models used as input the same set of features: the number of hops, number of spans, total path length (km), average link length (km), maximum link length (km), average span attenuation (dB), average dispersion (ps/nm/km) and type of modulation format.

It was concluded that, when predicting the lightpath feasibility, all the ML models had a good accuracy, above 90% in all cases (with the accuracy being defined as the ratio between the correctly predicted examples and the total number of examples). The ANN regression model did, however, present the best generalization capabilities, with accuracies above 99% even when training the model with smaller training sets. Furthermore, the ANN regression model was able to estimate the residual margin with an average error smaller than 0.4 dB.

In [27], the case of a multi-domain optical network, where multiple operators co-exist and the traffic can flow transparently through the different operator's domains, is considered. When a given optical channel traverses a domain which is not controlled by the operator which originated it, it is referred to as an alien wavelength. This paper proposes a solution based on a DNN which assesses the BER of an alien wavelength, assisting in the end-to-end multi-domain Routing, Modulation format, and Spectrum Allocation (RMSA) process.

This DNN model takes as inputs the source and destination nodes, data rate, physical length, modulation format and link occupation of an inter-domain optical channel, and outputs a flag indicating whether the BER of the lightpath satisfies the QoT requirement (it is a classification model). If it does satisfy the requirement then that connection is validated and set-up, otherwise, changes in the RMSA are made to guarantee the necessary QoT. The DNN model has 3 hidden layers with 10, 20 and 10 neurons respectively. The results showed that this DNN model achieves a prediction accuracy of 95.33%.

In [28], an ML solution is proposed to solve the Routing and Wavelength Assignment (RWA) problem in transparent optical backbone networks using WDM transmission. RWA consists in assigning a physical-path on the network and attributing wavelengths to all the optical channels associated to each

of the traffic demands in a given traffic matrix. The resolution of this problem can be done through ILP formulations that lead to optimal solutions. ILPs, however, suffer from high computational complexity, often requiring hours to solve medium-size network topologies. To alleviate this computational complexity, various heuristic algorithms have been proposed, offering faster computation times (typically separating this problem into a routing step and then a wavelength assignment step), but these lead to sub-optimal solutions.

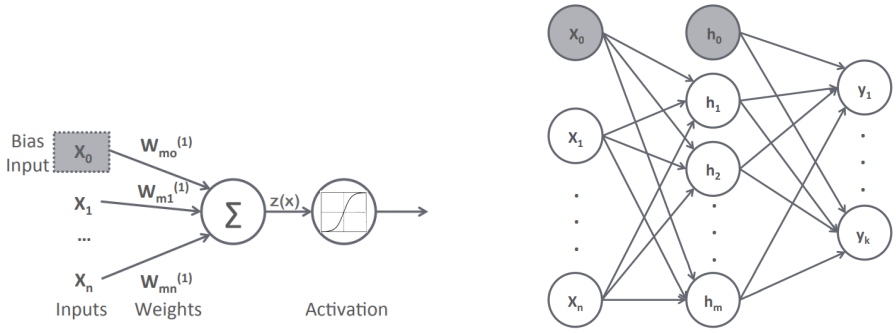
Given the problems present in these types of solutions, the authors of [28] chose to represent the RWA problem as a supervised classification problem. The input of a RWA problem is a serialized traffic matrix with all the source-destination traffic demands. The output is the Routing and Wavelength Configuration (RWC), a list in which, for each traffic demand (input), there is a sequence of links (the route chosen for that given traffic demand) and the wavelength assigned. This problem was transformed into a multi-class classification problem, where the outputs are predicted from a given set of RWC classes. Two ML algorithms were implemented: logistic regression and a DNN. Both ML algorithms output an array of values between 0 and 1 that can be interpreted as the probability for an RWC to be the correct solution for a given input. The RWC with the highest probability is then tested to see if it meets all the requirements of the source-destination RWA demands. If it doesn't, the RWC with the next highest probability is tested. This process is repeated for the top 10 RWCs.

It was concluded that the DNN model performs significantly better than the logistic regression model when it comes to predicting the correct class, especially in tests with a larger number of RWC classes. The DNN was also shown to be very good at replicating the results obtained by the ILP solution, resulting in a very high feasibility score, achieving almost complete feasibility (always above 95% in all the tests), while also reducing computation time by more than 93% with respect to classic ILP approaches.

The various articles analyzed in the context of the state of the art of machine learning techniques applied to optical networks underscores the prominence of neural networks models, showing that this type of models have consistently emerged as superior solutions in various applications within the field, providing high accuracy rates and fast computation times, reaffirming them as an optimal choice in the context of this thesis.

2.3.2 Theoretical Principles of Neural Networks

In this section, we delve into the theoretical principles that explain the operation of neural networks. Figure 2.6 provides a visual representation of a neural network, with Fig. 2.6(a) detailing the operations of a generic neuron m, and Fig. 2.6(b) illustrating the overall structure of a neural network with one hidden layer. Within each neuron, two key processes take place: the computation of the input activation (also known as pre-activation), and the subsequent generation of the neuron's output through an activation function.



(a) Detail of a generic neuron m.

(b) Neural Network.

Figure 2.6: Neural network with one hidden layer and detail of neuron operations. (Adapted from [5])

The input activation (pre-activation), represented by $z(x)$, is described by:

$$z(x) = \sum_{i=1}^{n} w_{mi} x_i + x_0, \tag{2.12}$$

where x_i are the input variables, w_{mi} are each of the connection weights and x_0 is the bias term.

The activation of the neuron is given by:

$$h_m(x) = g(z(x)), \tag{2.13}$$

where g is the activation function. Typical activation functions include: linear, sigmoid, hyperbolic tangent, rectified linear unit or softmax. In regression problems the Rectified Linear Unit (ReLU) ($g(z) = max(0, z)$) is commonly chosen as the activation function [29].

Figure 2.6(b) is a generic representation of a neural network with a single hidden layer, n inputs, m hidden units (the number of neurons on the hidden layer) and k outputs. Considering now the operation across the hidden layer (designated as layer 1), the hidden layer pre-activation is thus:

$$z^{(1)}(x) = W^{(1)} x + x_0, \tag{2.14}$$

with $W^{(1)}$ being a matrix (of size $m \times n$) containing the values of the connection weights between each of the inputs and each of the hidden units, x the vector of inputs and $x_0 \in \mathbb{R}^m$ the bias term (a vector of size m). Note also that $z^{(1)}(x)$ is a vector of size m.

The activation of the hidden layer is given by:

$$h^{(1)}(x) = g(z^{(1)}(x)), \tag{2.15}$$

with g being the activation function to be applied to each of the elements of vector $z^{(1)}(x)$. The vector

$h^{(1)}(x)$ is the output of hidden layer 1 and it will be fed into the next layer, which can be another hidden layer, or, in the case of Fig. 2.6, the output layer.

When it comes to the output layer, the same process as before is done, with a pre-activation, whose output can be designated $z^{out}(x)$ and an activation:

$$z^{out}(x) = W^{out}h^{(1)}(x) + h_0^{(1)}, \tag{2.16}$$

$$y(x) = o(z^{out}(x)), \tag{2.17}$$

where o is the output layer activation function. This function is typically the linear function ($o(z) = z$) in regression problems [5].

If more hidden layers were to be used (as in the case of DNNs), the expression that represents the hidden layer pre-activation for layer ℓ is:

$$z^{(\ell)}(x) = W^{(\ell)}h^{(\ell-1)}(x) + h_0^{(\ell)}, \tag{2.18}$$

with $W^{(\ell)}$ being a matrix (of size $m_\ell \times m_{\ell-1}$, m_ℓ being the number of hidden units of hidden layer ℓ) containing the values of the connection weights and $h_0^{(\ell)} \in \mathbb{R}^{m_\ell}$ the bias vector.

The activation of hidden layer ℓ is given by:

$$h^{(\ell)}(x) = g(z^{(\ell)}(x)) \ . \tag{2.19}$$

2.3.3 Training Neural Networks

The training of neural networks consists of determining the value for all of the neuron's weights (the values of the W matrices) and the bias values that minimize a given loss (or error) function with a given iterative method (optimizer algorithm).

In the training process, a dataset with features (inputs) and labels (outputs) is used, being split into a training set (the data used to determine the model's parameters), a validation set (used to make an unbiased evaluation of the model's performance during training) and a test set (used to assess the model's performance after the training is complete). Before the data is split into these three sets, the data needs to be pre-processed and shuffled. Data pre-processing consists in preparing the data to make it more suitable for the training process. This can be done by applying transformations to the data (to assure that all inputs are on the same scale through the standardization or normalization of the data, or to fix a skew in the distribution of a given feature or label), by filtering out given parts of the data (such as outliers, duplicates or invalid data), by aggregating features or creating new features from the existing ones [30].

The loss function is used to measure the difference between the value predicted by the Neural Network and the real value. That is, it measures the error associated to the model's predictions. In the case of regression problems the Mean Squared Error (MSE) is used [29]. The MSE is given by:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2, \tag{2.20}$$

where N is the number of data values being considered, $\hat{y}_i$ are the estimated values and y_i are the real values.

The optimizer algorithm is the method that determines how the weights and biases (the network parameters) are updated during the training process. Common optimizers include the Stochastic Gradient Descent (SGD) or the Adaptive Momentum Estimation (Adam). The update of the network parameters requires the computation of the gradient of the loss function, a task performed by the backpropagation algorithm [31]. The backpropagation algorithm is used in ANNs to calculate the gradient of the loss function with respect to the network's weights and biases. It does this by applying the chain rule of calculus and propagating the loss backward from the output layer to the input layer. The resulting gradients, which indicate the direction and rate of change of the loss function, guide the optimizer algorithm in iteratively adjusting the weights and biases.

An important parameter related to the optimizer is the learning rate. The learning rate determines the magnitude of the updates applied to the weights and biases during each iteration. A smaller learning rate implies smaller updates, requiring more iterations and thus a longer time for the model to converge. Conversely, a learning rate that is too large can lead to larger updates, which might cause the model to overshoot the global minimum of the loss function and fail to converge [29].

An epoch corresponds to a single iteration over the training data that results in an update of the neural network's parameters. In each epoch the training data is randomly divided into batches (that have a specific pre-defined size), meaning that between any two epochs, while the training data itself remains the same, the order in which the data is presented to the network changes. This is done to prevent the model from learning the order of the training examples and to ensure that the model generalizes well to unseen data. The following procedures occur in an epoch: the data is used to predict a set of outputs $\hat{y}_i$ (forward pass), the value of the loss between the predictions ($\hat{y}_i$) and the actual values (y_i) is computed both for the training set and the validation set, the backpropagation algorithm is done to compute the gradients, and, knowing the gradient values, the weights and biases of the model are updated. The training process will continue until a specified number of epochs is reached, unless Early Stopping is implemented.

Early Stopping is a technique used to prevent overfitting (a regularization technique) that consists in stopping the training process when the parameter updates no longer yield improvements on the validation set loss. Overfitting occurs when the model fits the training data too closely, including both real and

noisy data, which makes the model unable to generalize to new unseen data. By interrupting the training process before the validation loss starts increasing, overfitting can be avoided. The Early Stopping regularization can be adjusted by changing the patience parameter, which is a parameter that defines the number of epochs to wait for before interrupting the training. For example, if the patience parameter is set to 20, the model will only stop after 20 consecutive epochs with no improvement in the validation loss. This serves as a tolerance to the inherent fluctuations in model performance during training.

Another technique that is used to prevent overfitting is dropout regularization [32]. Dropout regularization consists in temporarily deactivating (dropping) random neurons from the neural network during the training process. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass and any weight updates are not applied to that neuron on the backward pass. The tuning of the dropout regularization can be done through the dropout probability, which defines the probability that a given neuron is dropped. The application of dropout regularization is done on a per batch basis, meaning that, for each batch of data being considered in a given epoch there is a new choice of neurons to drop. It was shown in [32] that dropout regularization can have a significant impact in reducing overfitting.

An important aspect of training a Neural Network is optimizing the hyperparameters. Hyperparameters are the variables that configure how the model learns from the data. This includes variables like the number of hidden layers, the number of hidden units, the learning rate, the batch size, the dropout probability, and others. During the training, different combinations of hyperparameters are tested to find the one that results in the best performance on the validation set. In this work, the performance of the model on the validation set is assessed using two metrics: the R^2 score and the Average Relative Error (ARE).

The R^2 score (R-squared), also known as coefficient of determination, is a statistical evaluation metric that indicates how well a model fits the data. In this context, it is used to compare the predictions made by the DNN model and the real values (determined through the routing process), being defined as [33]:

$$R^2 = 1 - \frac{\sum_{i=0}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{N}(y_i - \bar{y})^2}, \tag{2.21}$$

where y_i represents the actual value, $\hat{y}_i$ represents the predicted value, $\bar{y}$ is the mean of the actual values, and N is the number of data values being considered. The R^2 score will take values between 0 and 1, where a value of 1 indicates that the model fits the data perfectly and 0 that it does not fit the data at all. That means that the closer the values are to 1, the better the model is performing [33].

The ARE is the average of the relative errors between the predicted and actual values. A lower ARE indicates a better performance of the model on average [34]. It is defined as follows:

$$\text{ARE} = \frac{1}{N}\sum_{i=0}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| . \tag{2.22}$$

The relative error for each data point is calculated as:

$$\text{Relative Error} = \frac{y_i - \hat{y}_i}{y_i} . \tag{2.23}$$

While tuning a DNN model's architecture, another significant factor is the number of parameters. The number of parameters refers to the total count of the model's weights and biases. It is important to monitor this number because an overly large number of parameters can lead to overfitting. The determination of this value is influenced by the number of inputs, outputs, hidden units, and hidden layers, being given by:

$$\text{Number of Parameters} = \sum_{\ell=1}^{L}[(n_{\ell-1}+1)\times n_\ell] , \tag{2.24}$$

where L is the total number of layers in the network (including input and output layers), $n_{\ell-1}$ is the number of neurons in the $(\ell-1)$-th layer (or number of inputs), and n_ℓ is the number of neurons in the ℓ-th layer (or number of outputs). The term $(n_{\ell-1}+1)$ accounts for the weights connecting the neurons from the $(\ell-1)$-th layer to the ℓ-th layer and the additional "+1" accounts for the bias term for each neuron in the ℓ-th layer. The product $(n_{\ell-1}+1)\times n_\ell$ gives the total number of parameters between the $(\ell-1)$-th and ℓ-th layer. The summation over all layers gives the total number of parameters in the network. So, for example, considering a model with no hidden layers, having 12 inputs and 2 outputs, the total number of parameters would be: $(12+1)\times 2 = 26$. If there were 2 hidden layers with 50 hidden units each, then we would have: $(12+1)\times 50 + (50+1)\times 50 + (50+1)\times 2 = 3302$.

2.4 Chapter Conclusions

In this chapter, an overview of the fundamental concepts that serve as a basis for the work developed throughout this thesis was presented.

It was shown that telecommunication networks can be represented by their physical and logical topologies. The physical topology depicts the networks' physical layout and it can be represented as a graph or an adjacency matrix. The logical topology describes how the data is transmitted in the network, being represented by a demand matrix and traffic matrix. Various parameters related to both the physical and logical topology were detailed. Furthermore, the concept of routing, which is the process that defines the physical paths on the network through which the traffic will flow, was introduced, and a general strategy that can be applied to constrained and unconstrained routing problems was presented.

The concept of WDM transmission was explained, and it was shown that it allows for the concurrent

existence of multiple optical channels in the same optical fiber, operating in different wavelengths. However, given the bandwidth limitations inherent to fiber transmission, the number of optical channels that can simultaneously exist in an optical fiber link is limited. A way to determine the optical channel capacity, through the relation between optical reach and the Shannon channel capacity, was also presented. This strategy makes the process of determining the average channel capacity and total network capacity more straightforward and computationally less complex.

Various applications of ML techniques in the context of optical networks were analyzed and it was seen that the applications cover both physical-layer aspects (with the determination of QoT being a prevalent example) and network-layer aspects. Through the analysis of these applications it was concluded that ANN/DNN models tend to offer the best performances while also having very fast prediction times. For that reason, and given their universal approximation characteristic, the implementation of a DNN model was chosen for this thesis.

Chapter 3

Generation of Random Networks

In this chapter, the concept of random networks is introduced and the implementation of the generative graph model is detailed. The characteristics of the networks generated with this model are then analyzed to assure their similarity with real-world optical backbone networks.

3.1 Random Networks and Generative Models

The generation of random networks is a necessary step for the dissertation project, as reference networks (real-world networks often used in the context of scientific investigation) are not sufficient in number for the training and testing of the DNN models. These models will require large datasets which will be obtained from thousands of random networks. In that context, it is necessary that the generated networks have similar topologies to real-world optical backbone networks.

Random networks are networks whose edges (and in some cases the spatial position of the nodes) are random variables, that is, their physical topology is generated according to a given probability distribution, being represented by random graphs. This type of graphs was first studied by Paul Erdós and Alfred Rényi, who proposed a simple model for generating random networks, the Erdós-Rényi model [35].

In the Erdós-Rényi model, designated as $G(n, p)$, a graph is created by the random addition of links [35]. It takes as inputs the number of nodes (n) and the linking probability (p), which is the probability (with uniform distribution) that a link will be added between any pair of nodes, independently of other links. The resulting graph has a node degree with a Poisson distribution. Figure 3.1 shows a random network generated using the Erdós-Rényi model with $n = 15$ and $p = 0.5$.

This model is the simplest way of generating truly random networks, but, if the goal is to generate networks akin to real ones, total randomness is not always the best option, as real network's topologies have certain characteristics that are not taken into account in this model.

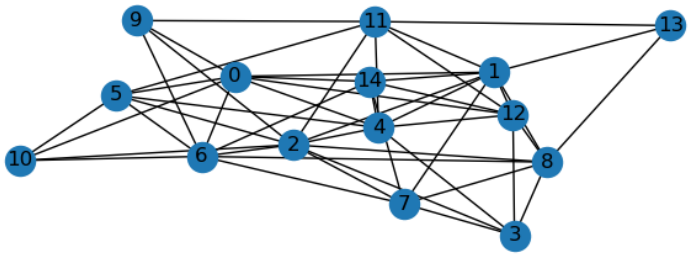


Figure 3.1: Random Graph generated with the $G(n, p)$ model with $n = 15$ and $p = 0.5$.

The Waxman model is another widely used generative graph model. This model considers the physical distance between nodes in the probability of node attachment, being thus designated as a geometric model [9]. This contrasts with the Erdós-Rényi model, which is a non-geometric model (where node positions and distances are not considered). In the Waxman model the nodes are placed uniformly at random in a two dimensional plane and the links are formed with a higher probability between nodes that are closer to one another. The probability that a node i establishes a link to node j is described by:

$$P(i, j) = \beta \exp \frac{-d(i, j)}{L_w \alpha}, \tag{3.1}$$

where $d(i, j)$ is the Euclidean distance between nodes i and j, L_w is the maximum distance between any two nodes, and α and β are parameters in the range of 0 to 1. Increasing β leads to an overall larger probability of links between any two nodes, while increasing α leads to a larger ratio of larger links to shorter links.

Assigning the nodes' positions in space makes the Waxman model better suited for describing realistic optical networks. However, this model still has some drawbacks, specifically, there is no guarantee that the generated topologies are connected, and that they are resilient to single link failures, both fundamental characteristics of optical backbone networks [9]. This type of resilience is defined by the edge connectivity $\lambda(G)$. A network being resilient to single link failures means that $\lambda(G) \geq 2$, meaning that between any pair of nodes there is always an alternative path in case there is a link failure.

Furthermore, the Waxman model brings other problems: connections between distant nodes are too common (even with smaller values of α and β) and the network's average node degree tends to excessive values when the number of nodes grows larger (which leads to networks with too many connections) [9].

3.2 Generating Realistic Optical Backbone Networks

To tackle these issues inherent to the Waxman model, a generative model with the specific goal of obtaining networks with topologies similar to real-world optical backbone networks was implemented.

This model is described in [9], but the implementation made in this dissertation project has some slight modifications (detailed in Section 3.2.1). This model, as the Waxman model, is also a geometric model where the nodes are randomly placed in a 2D plane and the Waxman connection probability, described by Eq. (3.1), is also used. The way the nodes are placed in the plane and the rules for establishing the connections between nodes are key differences between this model and the regular Waxman model.

In [9], an analysis of real-world optical backbone networks is made, where their most relevant characteristics are identified. It was observed that nodes are distributed based on the expected traffic in each geographic region (referring to any specified geographic area, which could be a city or a country), with some regions having more nodes than others and some having a single node or no nodes at all. In cases where a node is unique in a region, it tends to be connected to at least two nodes in neighbor regions, thus forming a cycle (which are important, as they provide survivability, giving each pair of nodes two disjoint interconnecting paths). In the case of two nodes in a region, a link usually exists between the two nodes and each of them is directly connected to at least another node in a neighbor region. Besides these general topology characteristics, a few other variables that characterize optical backbone networks were studied. The most relevant conclusions relative to these parameters were: when it comes to node degree, optical backbone networks tend to follow a Poisson distribution; the minimum node degree is 2 (a necessary, although not sufficient, condition for survivability) and all networks have at least two link-disjoint paths between each pair of nodes, $\lambda \geq 2$ (which indicates that the network is resilient to single link failures).

3.2.1 Description of the Implemented Model

The model implemented in this thesis was built on a software tool named *random_network*, which was developed by the author, making extensive use of the *Python* library *NetworkX*. The networks are created as *NetworkX* graph objects, which allows for a simpler manipulation of the networks (easily adding edges and giving them weights) as well as determining relevant network parameters (like the average node degree) and visualizing the graphs directly with the default *NetworkX* functions. At the end of the execution of *random_network* the generated networks are saved as NetworkX graphs into a single file.

This model has the following inputs: the number of nodes, N; minimum and maximum average node degrees ($\langle\delta_{min}\rangle$ and $\langle\delta_{max}\rangle$); the area of the plane A, through the specification of the side length L (the plane is a square, so $A = L^2$); the number of regions, R; the minimum distance between nodes, d; the α and β parameters of the Waxman link probability (in Eq. (3.1)); and the number of simulations (φ).

The model has the following general functioning: (1) dividing the plane into R regions; (2) placing a random number of nodes inside each region; (3) adding links between nodes inside each region; (4) adding links between nodes of different regions; (5) adding links with Waxman probability until $\langle\delta_{min}\rangle$ is reached; and (6) adding links with Waxman probability until $\langle\delta_{max}\rangle$ is reached, saving new networks

as the links are added. Every time a link is made a weight is assigned to it, corresponding to the distance between the nodes in the plane. Figure 3.2 is the flow diagram from [9] that describes this model.

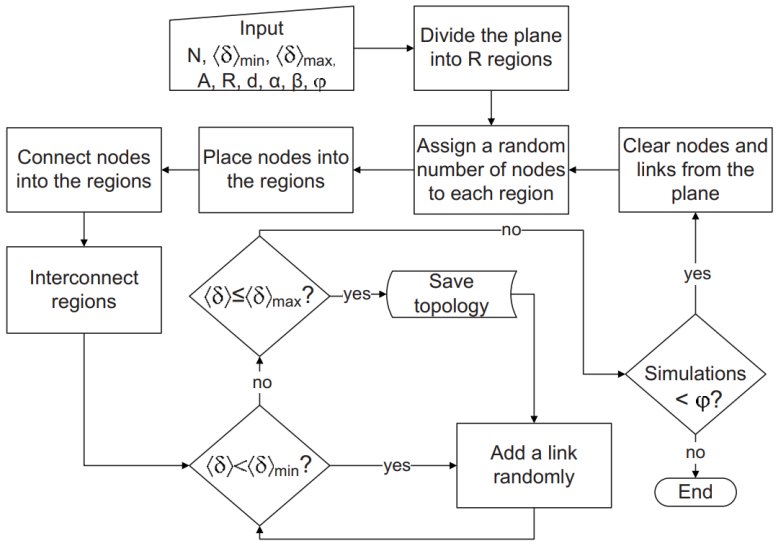


Figure 3.2: Flow diagram of the generative model. (From [9])

After step 4, the network (defined as a graph *G*) should have a connected topology that is survivable to single link failures. For that reason, the connections established between the nodes, either nodes inside the same region or nodes of different regions, needs to be done according to specific rules.

When connecting nodes inside a given region (step 3), if a region has two nodes, they are directly connected; if there are three nodes or more, they are connected as a cycle according to their relative angular positions with respect to the centroid of their locations. The nodes are sorted based on the angles they make with the centroid, and each node is then connected to its next and previous node in this sorted list, forming a cycle.

The implementation of this type of connection is a key difference as to what was described in [9]. In [9] when there are three nodes in a region, these nodes are connected in a cycle, but for regions with more than three nodes, the connections are made following the Waxman link probability. The application of the Waxman probability in the intra-region connections in these situations tends to lead, however, to the establishment of more connections than necessary (given the random nature of these connections and the fact that enough connections need to be added until the minimum node degree of 2 is reached), making it so that the specified $\langle \delta_{min} \rangle$ was often exceeded by a significant margin before the additional links were added in step 5. Furthermore, the topologies tended to have a less realistic appearance as these connections gave the intra-region connections a more random topology (where, for example, distant nodes in the region had links while closer ones did not).

Connecting the regions with 3 nodes or more based on their relative positions to the centroid of their

locations proved to be a better solution, that guarantees the minimum node degree of 2 to all nodes while also providing a good structure over which more connections can be added.

After having all the nodes inside the regions connected, the inter-region connections are made (step 4). In [9] these connections are established according to the Waxman link probability between links of different regions. However, this type of solution had the tendency of establishing links between very distant nodes (even after adjusting the α and β parameters) resulting in unrealistic topologies. Instead, in the algorithm implemented by the author, the inter-region connections are established by creating an auxiliary graph G_2 where each node represents a region in the original graph G, and the weight of an edge between two nodes is the distance between the corresponding regions. A Minimum Spanning Tree (MST) algorithm is applied to G_2 to connect all the nodes (regions) with the minimum total edge weight. An MST is a subset of the edges of a graph that connects all the nodes together, without any cycles and with the minimum possible total edge weight. Using an MST in this context ensures that all regions are connected while minimizing the total distance between them, ensuring that the closest regions are the ones where the connections are established.

The edges of the MST are then used to connect the regions in G, with each edge in the MST corresponding to two links between each of the considered regions in G. For each pair of regions connected by an edge in the MST, the closest pair of nodes, one from each region, are connected. If a region has more than one node, a different node from the region is used for the second link. This ensures that there are at least two link-disjoint paths between any two regions, enhancing the resilience of the network.

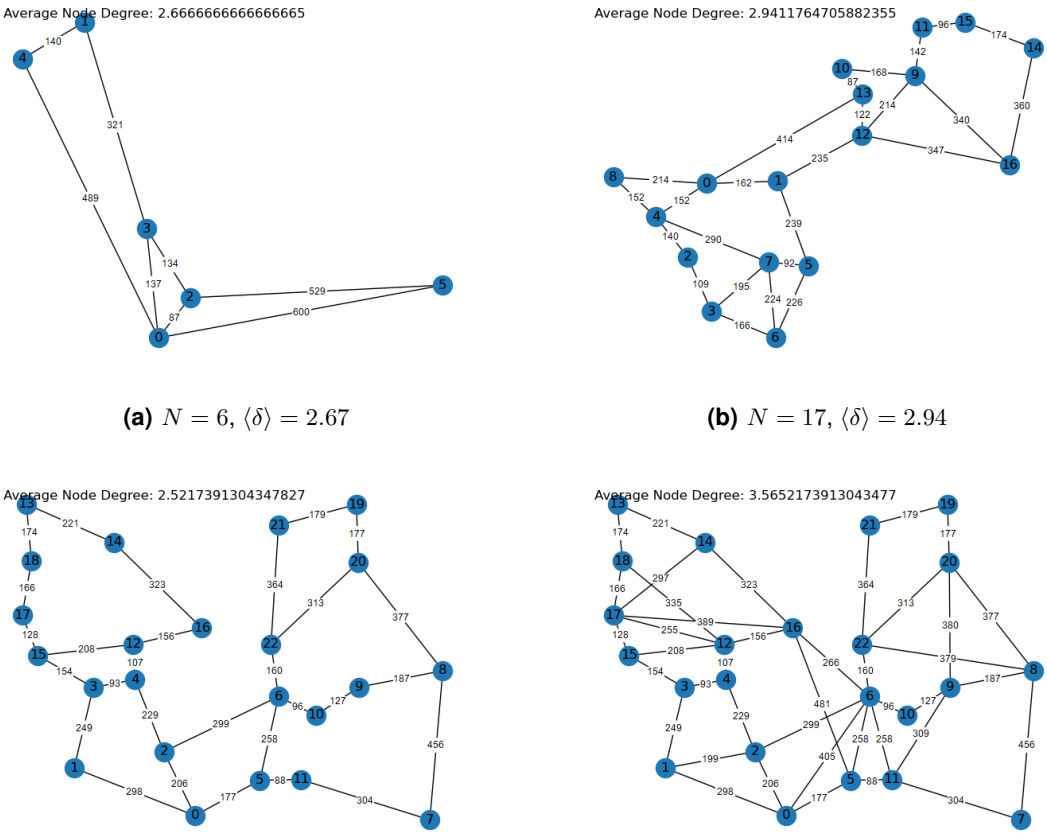
After this point, all the nodes have a minimum degree of 2 and the topology is connected. Also, the network does not have any bridge (or cut-edge), that is, edges whose removal would disconnect the graph, making the topology resilient against single link failures, with $\lambda \geq 2$, as intended. Note that this strategy also tends to generate networks with at least two node-disjoint paths (resilient to single node failures) very frequently, but this is not guaranteed, as regions with just one node exist and this node can be critical in the paths between other neighbor regions. While this type of resilience is an advantage for optical backbone networks, resilience to node failures is not a characteristic that all real-world optical backbone networks have (as was concluded in [9]) and so this requirement was not considered as essential in this implementation.

In the next steps, all the links added to the network are established between any pair of nodes in the network according to the Waxman probability. In a first phase this is done until the $\langle \delta_{min} \rangle$ is reached (step 5) and then until $\langle \delta_{max} \rangle$ (step 6). In step 6, as new links are added between $\langle \delta_{min} \rangle$ and $\langle \delta_{max} \rangle$, multiple network topologies are saved. This allows for the obtainment of several network topologies with the same node distribution, but with different average nodal degrees. In [9] it is proposed that the saving of a new network occurs for every new link that is added. However, to guarantee that the datasets obtained are comprised of a more varied set of networks, the saving of a new network in the

implemented model is done based on the variation of the average node degree instead. To do that a parameter, $\Delta\delta$, can be defined. This parameter makes the saving of a new network occur only after the average node degree changes by at least that value, assuring that the difference between each saved topology can be adjusted according to what is needed.

3.2.2 Results of the Implemented Model

Figure 3.3 shows some examples of networks generated with *random_network*. The networks in Fig. 3.3(c) and Fig. 3.3(d) were generated from the same node distribution.



(a) $N = 6$, $\langle\delta\rangle = 2.67$

(b) $N = 17$, $\langle\delta\rangle = 2.94$

(c) $N = 23$, $\langle\delta\rangle = 2.52$

(d) $N = 23$, $\langle\delta\rangle = 3.57$

Figure 3.3: Examples of networks obtained with *random_network*.

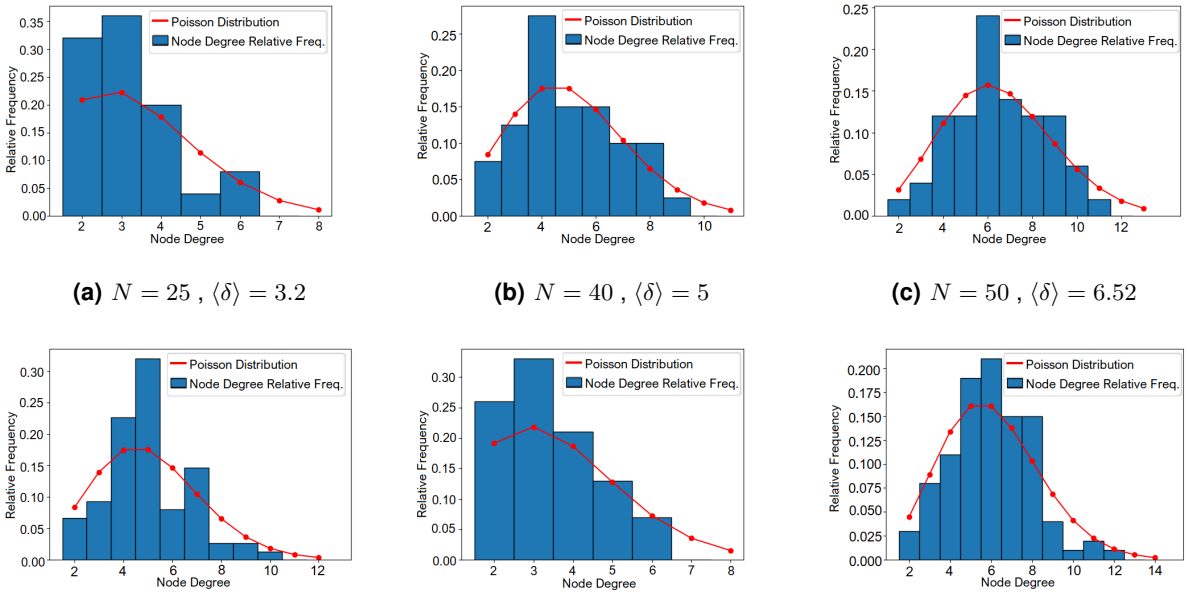
The examples in Fig. 3.3 make it evident that the generated networks have connected topologies with an edge connectivity of at least 2 (resilient to single link failures) and that the connections are more prevalent among nodes that are closer together. The existence of separate regions can also be noticed, particularly in Fig. 3.3(a) and Fig. 3.3(b). For example, in Fig. 3.3(b) two regions can be seen: one on the bottom left (with nodes 0 through 8), and other on the top right (with nodes 9 through 16).

To ensure that the generated networks are similar to real optical backbone networks, it is important to compare certain statistics. In [9] it was concluded that the node degree of such networks follows a Poisson distribution. The following expression is the probability mass function of the Poisson distribution:

$$p(k) = \frac{\mu^k}{k!} e^{-\mu}, \text{with } k = 1, 2, \dots , \tag{3.2}$$

where k is the number of events and μ is the average number of events. Saying that the the node degree of a network follows a Poisson distribution means that μ is the average node degree ($\mu = \langle\delta\rangle$) and that Eq. (3.2) describes the probability of a given node having exactly k link connections.

The node degree distribution of 6 random networks obtained with *random_network* (with a varying number of nodes and average node degree) was determined and the results are depicted in Fig. 3.4.



(a) $N = 25$, $\langle\delta\rangle = 3.2$ **(b)** $N = 40$, $\langle\delta\rangle = 5$ **(c)** $N = 50$, $\langle\delta\rangle = 6.52$

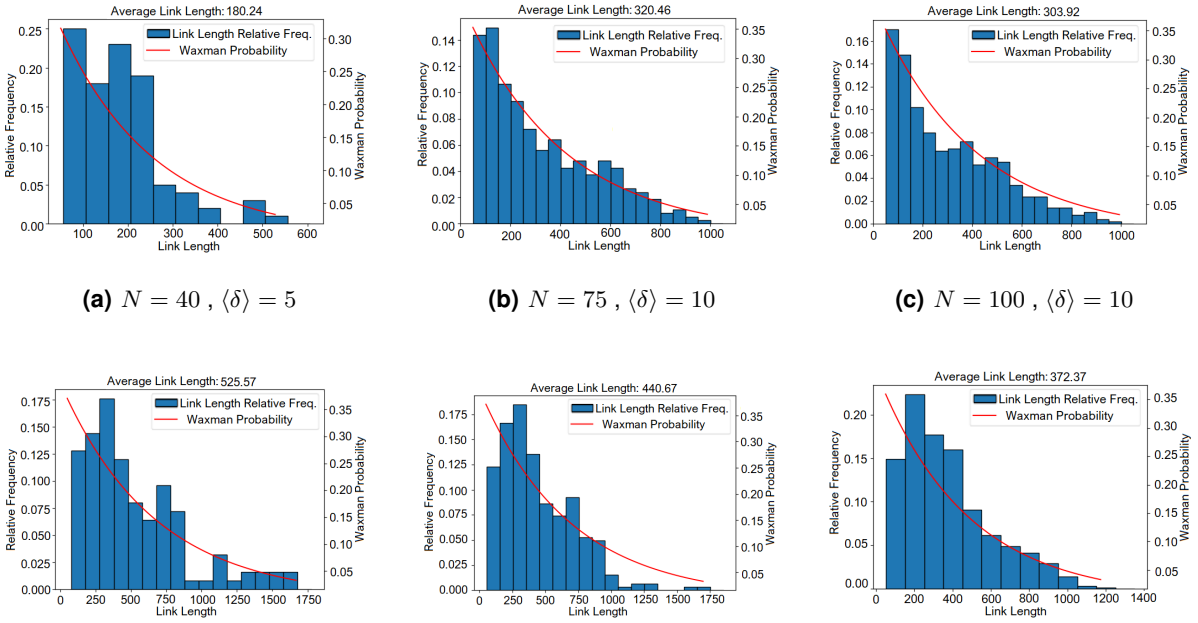
(d) $N = 75$, $\langle\delta\rangle = 5.01$ **(e)** $N = 100$, $\langle\delta\rangle = 3.42$ **(f)** $N = 100$, $\langle\delta\rangle = 6$

Figure 3.4: Node degree distribution for 6 random networks compared with the Poisson distribution with $\mu = \langle\delta\rangle$.

Figure 3.4 shows that the generated networks tend to follow a Poisson distribution, especially when the average node degree is higher. It can also be seen that the minimum node degree is 2 in all cases, which is an intended characteristic, as it is a necessary condition to guarantee networks with topologies survivable to single link failures.

In [9], a similar analysis was made for the *USA 100* network, which is a real-world network with $N = 100$ and $\langle\delta\rangle = 3.42$. The obtained plot for that network is very similar to the one in Fig. 3.4(e), which has the same number of nodes and average node degree (despite the topology being different). This similarity further confirms the correct node degree distribution of the generated networks.

When it comes to link length, optical backbone networks tend follow the Waxman link probability with $\alpha = \beta = 0.4$ [9]. Figure 3.5 has the plots for the link length distribution of 6 networks, generated considering these Waxman parameters and a varying number of nodes and node degree.



(a) $N = 40$, $\langle \delta \rangle = 5$ **(b)** $N = 75$, $\langle \delta \rangle = 10$ **(c)** $N = 100$, $\langle \delta \rangle = 10$

(d) $N = 50$, $\langle \delta \rangle = 5$ **(e)** $N = 100$, $\langle \delta \rangle = 6.5$ **(f)** $N = 200$, $\langle \delta \rangle = 6.5$

Figure 3.5: Link Length distribution for 6 random networks and the Waxman probability with $\alpha = \beta = 0.4$.

From Fig. 3.5, it can be seen that the link length tends to follow the Waxman probability distribution, with shorter link lengths being more common than longer ones in all cases.

3.3 Chapter Conclusions

This chapter introduced the concept of random networks and presented two generative models: Erdós-Rényi and Waxman. Despite the Waxman model's potential for creating networks akin to optical backbone networks (given that it is a geometric model), its lack of guaranteed minimum edge connectivity of 2 and connected topologies limit its suitability, given these are key features of such networks.

The model developed by the author of this work (based on [9]), which makes use of the Waxman link probability, was designed to ensure these features. It was concluded that the characteristics of the generated networks have a close resemblance to real-world optical backbone networks, both in terms of node degree distribution, link length distribution, as well as the resilience to single link failures.

Chapter 4

Routing, Wavelength Assignment and Fiber Assignment

In this chapter, the implementation of the routing algorithms is detailed and the results of their application to reference networks and artificially generated networks are presented. Two routing algorithms are presented: a constrained routing algorithm, where the limitation in the number of optical channels per fiber in DWDM transmission is considered and there is the possibility of traffic demands being blocked; and an unconstrained routing algorithm with fiber assignment, where the limitation in the number of optical channels per fiber is overcome through the addition of optical fibers to the network's links, which corresponds to an SDM solution.

4.1 Constrained Routing

As defined in Section 2.1.3, routing is the process of determining the path ($\pi_{s,d}$) in the physical topology of a network for each traffic demand ($t_{s,d}$) between the respective source node (s) and destination node (d). As there are various possible paths between the various nodes, the objective is to find the shortest path, that is, the path that minimizes the total path length $l(\pi_{s,d})$. This can be achieved using an heuristic algorithm such as Dijkstra's algorithm.

In the case of constrained routing, this process also needs to account for the links' capacity, that is, the limitation on the number of optical channels that can traverse a given link at the same time, assuming DWDM transmission. Considering that each link only has a single optical fiber per communication direction, the link capacity is based on the transmission bandwidth and the channel spacing, as explained in Section 2.2.1. The goal of the constrained routing algorithm is to maximize the number of allocated traffic demands (minimizing the blocking ratio) given the restricted link capacity.

During the routing process, each path $\pi_{s,d}$ is physically established with an optical channel and so, it is also during this process that the wavelength for each optical channel is attributed. This is called wavelength assignment. For that reason, the implemented routing algorithm is an heuristic algorithm used to solve the RWA problem, with the main objective being, for each optical channel k (corresponding to each traffic demand), the definition of a given path π_k and wavelength λ_k.

The process of wavelength assignment was done according to the first-fit strategy [36], being subject to the following constraints: (a) wavelength continuity, meaning that all the links in the path of an optical channel must use the same wavelength (and therefore each optical channel is associated to a single wavelength); and (b) each link must carry unique wavelengths, that is, if multiple optical channels pass through the same link, they must each have a different wavelength to avoid interference.

Besides the RWA solution, through which the network capacity and average channel capacity are determined (the labels of the DNN model), the implemented algorithm is also used to determine the set of network parameters used as the DNN's inputs. Therefore, this algorithm also outputs various parameters related to a network's physical topology.

4.1.1 Inputs and Outputs of the Program

The program developed to implement the constrained routing algorithm was named *routing_nx*. This program takes the following inputs: a *gpickle* file containing a list of *NetworkX* Graphs [7] (which is a serialized byte stream of the Graph objects that preserves the graph's node positions and link lengths), the value of the maximum number of optical channels per link (link capacity), and the sorting strategy used to define the order the traffic demands will be routed (shortest-first, longest-first or largest-first). Optionally, the user can input a traffic matrix $T = [t_{s,d}]$. If it is not specified, a matrix for a full-mesh logical topology with one unit of traffic is assumed by default. This traffic matrix can be defined as:

$$t_{s,d} = \begin{cases} 1, & \text{if } s \neq d \\ 0, & \text{if } s = d. \end{cases} \tag{4.1}$$

Note that the input *gpickle* file is the same type of file output by the program *random_network* (in Section 3.2.1), as the idea is to load the randomly generated networks directly into *routing_nx*. However, this file does not necessarily need to be originated from *random_network*. As long as the correct format is input, this program accepts any *gpickle* file with a list of *NetworkX* graphs, including lists with a single graph. This flexibility allows the use of *routing_nx* on a variety of networks, including reference networks.

Note also that the ability of defining the link capacity allows for the specification of different transmission scenarios (with different WDM bandwidths and channel spacings). By setting the link capacity to a very high value (e.g. 99999), this program can be used to solve unconstrained routing problems.

At the end of its execution, *routing_nx* outputs a *.npy* file that contains a 2D *numpy* array (matrix) with

the determined data. This output array can then be directly loaded into the code that implements the DNN algorithm's training (see Chapter 5). Each of the output matrix's rows has the data relative to an analyzed network (each row is an example from the perspective of the DNN training) and each column has a different network parameter (feature), except the last two columns which contain the total network capacity and the average channel capacity (the labels in the first DNN model). When training the DNN, the labels should be considered in a separate vector, but for saving purposes, all this data is considered as a single matrix.

This program is capable of determining a total of 23 network parameters, although only 12 parameters are saved to be used in the DNN. The parameters which are saved are: number of nodes, number of links, minimum, maximum and average link length, variance of link length, minimum, maximum and average node degree, variance of node degree, network diameter, and algebraic connectivity. Since there is a total of 12 parameters saved, together with the labels, each row of the output matrix has 14 elements. The output matrix's dimensions are then ($N_{nets} \times 14$), where N_{nets} is the number of networks analyzed. The remaining parameters were not included in the DNN's inputs as the goal is to have a DNN model which is able to make predictions based solely on the network's physical characteristics and these remaining parameters require the routing solution to be determined. These are: average number of hops per demand, number of paths, minimum, maximum and average path length, average number of hops per path, minimum, maximum and average link load, number of blocked traffic demands, and blocking probability.

Besides these parameters *routing_nx* also outputs: the load in each link; the wavelengths of the optical channels present in each link; the paths chosen for each traffic demand, π_k, and their lengths, $l(\pi_k)$; the wavelengths associated to each traffic demand, λ_k; and a list of the blocked traffic demands and the amount of blocked traffic.

4.1.2 General Functioning of the Program

The general workings of *routing_nx*, defined by its main function, can be explained by Algorithm 1. Here, after the parsing of the command-line arguments (used as described in Appendix A.1) and the loading of the file with the list of *NetworkX* graphs, the main loop is started. In this loop one network is analyzed at a time (each network being a *NetworkX* graph, *G*, which is a weighted graph with the link lengths, as described in Section 2.1.1) and the network's parameters and routing solution are determined. The results are appended to a matrix, *dnn_data*, which after all the iterations of the main loop will have the shape $N_{nets} \times 14$. The results can also be printed in the terminal, as shown in Appendix A.2.

The main steps of *routing_nx*, as described in Algorithm 1, are: (a) the determination of the shortest-paths between all pairs of nodes (Line 8); (b) the ordering of the traffic demands according to a specified sorting strategy (Line 10); (c) the routing of the traffic demands following the specified order, including

the determination of the paths π_k and the wavelengths λ_k (Line 11); and (d) the calculation of the average channel capacity and total network capacity (Line 13) according to Eq. (2.4) and Eq. (2.7), with the capacity of each optical channel being determined through the optical reach, as explained in Section 2.2.2.

Algorithm 1: Main Function (General functioning of *routing_nx*)

Input: A *.gpickle* file with a list of *NetworkX* Graphs, the sorting strategy, *sorting_order*, and the maximum link load, *MAX_LINK_CAP*.
Output: A file with a 2D *numpy* array with the determined parameters, *dnn_data.npy*.

```
1:  Parse the input-line arguments
2:  Load the data from the file
3:  for each network G in loaded networks do
4:      if G is not connected OR number of nodes is less than 3 then
5:          Skip to the next iteration of the loop
6:      if traffic matrix not defined OR traffic matrix has wrong dimensions then
7:          Create a traffic matrix for a logical full-mesh topology
8:      Call function to determine shortest-paths between all pairs of nodes (See Alg. 4 in App. A.3.1)
9:      Calculate the parameters related to physical and logical topology and the hop_matrix
10:     Call orderPaths to sort the traffic demands according to sorting_order (See Alg. 5 in App. A.3.2)
11:     Call function to route the traffic and assign wavelengths (See Algorithm 2)
12:     Calculate the network parameters based on the routing solution
13:     Determine the average channel capacity and total network capacity
14:     if PRINT_MODE is True then
15:         Print the results in the terminal (Example in Appendix A.2)
16:     else
17:         Append all the determined parameters to dnn_data
18: if PRINT_MODE is False AND dnn_data is not empty then
19:     Convert dnn_data to a 2D numpy array and assign it to dnn_data_np
20:     Call function to save dnn_data_np as a .npy file
```

The function for finding the shortest paths between each pair of nodes (Alg. 4) is described in Appendix A.3.1. This process makes use of the *NetworkX* function *all_shortest_paths*, a function that implements *Dijkstra*'s shortest-paths algorithm in an efficient way, having the advantage of returning all shortest paths if multiple paths of the same length exist between a pair of nodes, which is a desired characteristic for this algorithm, as these multiple paths are accounted for in the routing.

After having the list of the shortest paths, the traffic demands are ordered according to the specified sorting order, as described in Section 2.1.3, with the shortest-first and longest-first strategies being based on path length. The algorithm that explains this process (Alg. 5) is presented in Appendix A.3.2.

The output of this process will be a list with the ordered traffic demands, each traffic demand being represented by a dictionary that contains the keys: (a) 'source': the source node; (b) 'destination': the target node; (c) 'distance': the total length of the path; (d) 'path': a list of all shortest paths between the source and target node; (e) 'traffic': the traffic between the source and target nodes, as given by the traffic matrix; (f) 'hops': the number of hops between the source and target nodes; and (g) 'routed': a boolean value indicating whether the traffic demand has been routed (set to False in this function).

4.1.3 Routing and Wavelength Assignment Process

The function *route* is used to determine the routing solution. This function takes as input the list of ordered traffic demands and a copy of *G* (the current graph being analyzed), G_{copy}, to which changes can be made in case of link saturation (removing links). The *route* function is detailed in Algorithm 2:

Algorithm 2: Route Function (Routing and Wavelength Assignment)

Input: The list of ordered traffic demands, *ordered_traffic_demands*, and a copy of *G* (the current network's graph), named G_{copy}.
Output: Matrices and variables that define the RWA solution: *load_matrix*, *path_matrix*, *distance_matrix*, *blocked_traffic*, *blocked_paths*, *path_wavelength_matrix*, and *link_wavelength_matrix*.

```
Initialize matrices and variables
new_paths_flag ← False                ▷ Is set to True when network is updated
for each p in ordered_traffic_demands do ▷ Each p is the dictionary output in orderPaths. Key 'path' has a list of all possible paths. If more than one is present, a tie occurs.
    if p is already routed then
        Continue to next path
    if new_paths_flag is True then
        p ← new_shortest_path ▷ If at any point the network is ever updated, the possible paths are now considered from the new shortest-paths
    if p has no available paths to destination (is blocked) then
        Update blocked_traffic and blocked_paths
        Continue to next traffic demand
    if there is only a possible path then
        chosen_path ← that path's index in p['path'] list
    else
        chosen_path ← breakTie (see Algorithm 6)
    Call assign_wavelengths and then route_path (See Algorithms 7 and 8)
    if there are no available wavelengths OR blocked_flag is TRUE then
        Update blocked_traffic and blocked_paths
        Continue to next traffic demand
    for each path p2 in ordered_traffic_demands with p2['source'] = p['destination'] AND
    p2['destination'] = p['source'] AND p_aux['traffic'] = p2['traffic'] do
        Choose the same path index as p   ▷ Ensure connection d-s is done the same as s-d
        Call assign_wavelengths and then route_path (See Algorithms 7 and 8)
    if network needs to be updated then ▷ Set in route_path when a link reaches residual capacity 0
        new_shortest_paths ← Call update_network (see Algorithm 9)
        new_paths_flag ← True
return load_matrix, path_matrix, distance_matrix, blocked_traffic, blocked_paths,
path_wavelength_matrix, link_wavelength_matrix
```

The *route* function has seven outputs: *load_matrix*, *path_matrix*, *distance_matrix*, *blocked_traffic*, *blocked_paths*, *path_wavelength_matrix*, and *link_wavelength_matrix*. These outputs structures represent the RWA solution: (a) the *load_matrix* is a matrix of size $N \times N$ (N being the number of nodes) that contains, in position (i, j), the total load in the link between nodes i and j (as defined in Section 2.1.2); (b) the *path_matrix* ($N \times N$) which has, in position (i, j), the path chosen to route the traffic demand between node i and node j (represented as a sequence of nodes); (c) the *distance_matrix* ($N \times N$) that contains,

in position (i, j), the length of that given path between nodes i and j (this length will not necessarily be the same as the original shortest-path distance used to sort the traffic demands, since, in the case of constrained routing, the chosen path may not be the original shortest path); (d) the variable *blocked_traffic* that represents the amount of blocked traffic demands; (e) the list *blocked_paths* that contains the paths corresponding to the traffic demands which were blocked; (f) the matrix *path_wavelength_matrix* ($N \times N$) that has, in position (i, j), the wavelength attributed to the optical channel between nodes i and j; and (g) the matrix *link_wavelength_matrix* that has, in position (i, j), a list of all the wavelengths present in the link between nodes i and j.

The routing process, as explained in Algorithm 2, consists of a main loop which iterates over each traffic demand in the ordered list of traffic demands. When there are multiple shortest paths with the same length for a given traffic demand, the first step in the routing process is the decision of the path to route the traffic through. This choice is done in the *breakTie* function (described in Appendix A.3.3). This function returns the chosen path based on the principle of minimizing the load in the most loaded link, considering all the possible shortest paths.

After having chosen a path, the next process is the wavelength assignment, in the function *assign_wavelengths* (described in Appendix A.3.4). This function implements the first-fit wavelength assignment strategy [36], subject to the constraints described in Section 4.1 (wavelength continuity and the impossibility of equal wavelengths existing on the same link). In this process, the wavelengths being assigned are simply represented as integers starting in 1 and up to the maximum number of optical channels per fiber. After the execution of this function, if the process was successful (able to assign a wavelength to the optical channel, given the aforementioned constraints) this function returns the boolean value *True* to the *route* function and the *path_wavelength_matrix* and *link_wavelength_matrix* are updated accordingly. If it was not possible to find a wavelength that satisfies the necessary requirements, the boolean value *False* is returned to the *route* function, indicating that traffic demand needs to be blocked (due to wavelength unavailability).

If the wavelength assignment process was successful, then the *load_matrix* and *path_matrix* need to be updated accordingly. This is done in the *route_path* function (described in Appendix A.3.5). Besides updating these matrices, this function also verifies if any of the links has reached residual capacity zero. If that is the case, a flag variable *update_net_flag* is set to *True* to indicate that the network needs updating, and the saturated links are added to a list of links to be removed.

At the end of the main loop of the *route* function, in case the *update_net_flag* is set to *True*, the *update_network* function (described in Appendix A.3.6) is called. This function updates the auxiliary graph G_{copy}, by removing the congested links, and then it recalculates the shortest paths through the same method as used before. From this point on, and every time the network is updated and new shortest paths calculated, the routing of the remaining traffic demands considers only the new set of

paths. This type of behavior allows for the routing algorithm to explore all the possible paths between any pair of nodes. So, as the links in the network start getting saturated and the original paths start becoming unavailable, new alternative paths are found until no more alternative paths exist.

Note also that, by design, the routing of "symmetric" traffic demands is done contiguously and through the corresponding reverse path whenever the input traffic matrix is symmetric. That means that the demand from node j to node i is routed immediately after the demand from node i to node j, and that if the path from i to j was $[i, a, b, j]$, the path from j to i will be $[j, b, a, i]$.

In Appendix A.4, a simplified example of the application of the constrained routing algorithm to a small test network is presented, with the objective of illustrating the process of defining a path π_k in a constrained routing scenario.

4.2 Unconstrained Routing with Fiber Assignment

In the constrained routing scenario, described in Section 4.1, there was a possibility of traffic demands being blocked due to the limited number of optical channels per link and the restrictions in the process of wavelength assignment. In networks with a large amount of traffic (in full-mesh logical topologies being the ones with more nodes) this approach can lead to a large number of blocked traffic demands.

Optical backbone networks are typically designed to avoid the blocking of traffic demands. This can be achieved, for example, through the use of SDM, that is to say, through the deployment of additional optical fibers to the network's links. The addition of enough fibers to the network allows for the reduction and eventual elimination of the blocking, as each fiber added to a link increases that link's capacity. The process of determining which links should receive additional fibers, and how many fibers to add, is referred to as fiber assignment. The determination of a fiber assignment solution that results in a minimization (or elimination) of the blocking while also minimizing the amount of deployed fiber can be, however, a complex procedure that requires long computation times [37].

For that reason, the fiber assignment solution proposed in this work has the goal of completely eliminating the traffic demand blockages while also prioritizing reduced computation times, even if it means somewhat sacrificing the efficiency of fiber placement. The necessity for low computation times is critical, as the goal of this routing algorithm is to determine routing solutions for thousands of networks, with the determined results being subsequently used to train a DNN model. This second DNN model has the objective of predicting the unconstrained network capacity (as given by Eq. (2.7) where N_{blocked} is zero) and the total cost in kilometers of deployed fiber.

As the goal is to completely avoid the blocking of traffic demands, a strategy of adding fibers to the links "as needed" was followed. That is, every time a given link cannot accommodate a given traffic demand, an additional fiber is activated. To achieve this strategy, the routing can be done in an unconstrained manner, letting the traffic demands follow their originally determined shortest paths and

not imposing a limit on the number of optical channels per link. After the routing, knowing the maximum number of optical channels than can exist in a fiber (as defined for the link capacity) and the wavelengths in each link, the number of optical fibers that need to be added to each link is then determined.

This strategy led to the development of an alternative version of the program *routing_nx* which was named *routing_nx_no_blocking*. This new version of the program is identical in almost all aspects to the previous version (being also generally described by the same algorithms explained in Section 4.1) with the only differences being that now there is no limit to the number of optical channels per link (*MAX_LINK_CAP* is set to 99999), and after the *route* function (Line 11 in Algorithm 1) the number of fibers per link is determined, as well as the cost in kilometers of fiber.

Since there is no blocking of traffic demands when using this approach, the blocking related outputs (*blocked_traffic* and *blocked_paths*) are no longer present. Instead there are now new outputs: (a) *fiber_link_matrix*, a matrix of size $N \times N$, where the element in position (i, j) has the number of fibers used in the link between nodes i and j; and (b) *total_cost*, the total number of kilometers of fiber in the network. This last output is given by:

$$\Lambda_{\text{net}} = \sum_{i,j} l_{i,j} \times nf_{i,j} \ , \tag{4.2}$$

where $l_{i,j}$ is the length of the link between the nodes i and j (in km), and $nf_{i,j}$ is the number of optical fibers in the link between nodes i and j, as represented in the respective position of the *fiber_link_matrix*.

Other parameters such as the minimum, maximum and average number of fibers per link, as well as the average cost per link (in km of fiber) can also be determined using this program.

The function used to determine the number of optical fibers in each link, which is used after the process of unconstrained routing (*route* function), is called *assign_fibers*, and it is described by Algorithm 3.

Algorithm 3: Fiber Assignment

Input: the maximum number of wavelengths per fiber (*max_wavelengths*), the matrix with a list of the wavelengths in each link (*link_wavelength_matrix*), and the NetworkX graph, G.
Output: a 2D numpy array that represents the number of fibers in each link *fiber_link_matrix*.

```
1:  Initialize empty numpy array fiber_link_matrix.
2:  for i in lines of link_wavelength_matrix do
3:      for j in columns of link_wavelength_matrix do        ▷ Iterate over the link_wavelength_matrix.
4:          if G has an edge i, j then
5:              if there are no wavelengths in link_wavelength_matrix[i][j] then
6:                  fiber_link_matrix[i][j] ← 1              ▷ Even with no traffic, there is a link with one fiber
7:              else
8:                  normalized_wavelengths ← link_wavelength_matrix[i][j] mapped in the range 1 to
                    max_wavelengths
9:                  num_fibers ← maximum number of wavelength repetitions in normalized_wavelengths
10:                 fiber_link_matrix[i][j] ← num_fibers
11:         else                                             ▷ Case there is no edge i,j
12:             fiber_link_matrix[i][j] ← 0
    return fiber_link_matrix
```

The inputs of this function are: the maximum number of wavelengths per fiber (*max_wavelengths*), the *link_wavelength_matrix* (output of the *route* function), and the *NetworkX* graph, *G*. It outputs the *fiber_link_matrix*. Note that in this matrix the number of fibers is presented by direction, that is, the number of fibers in the link from i to j separately from the number of fibers in the link from j to i (each in the respective positions of the matrix). Given the way the routing is done, this matrix (as well as the *link_wavelength_matrix*) is symmetric when the traffic matrix is symmetric.

In the *assign_fibers* function, each link is analyzed individually in a loop. The number of fibers is then determined based on the list of wavelengths associated to that link. If the link does not exist (determined through the graph *G*), the number of fibers is set to zero; if the link exists but there are no wavelengths (meaning there is no traffic in that link), the number of fibers is set to one (despite there not being any traffic, the link still physically exists in the network, so a fiber needs to be assigned); if a link exists and has traffic (the list of wavelengths is not empty), then the number of fibers is determined in lines 8 to 10 of Algorithm 3. In this part of the algorithm two steps are made: the normalization of the wavelengths of that link, and then the counting of the number of repeated wavelengths and finding of the maximum, which will determine the number of fibers needed for that link.

When the routing is made in an unconstrained manner, the number of wavelengths in a link will not have a limit. There is not, however, an unlimited number of wavelengths to assign to the traffic demands; that limit is defined by *max_wavelengths*, which corresponds to the maximum number of optical channels in an optical fiber. That means that the list of wavelengths in a link, which is a list of integers without repetitions ranging from 1 to $+\infty$ (resulting from the first-fit strategy done in the *route* function, and being represented in the respective position of the *link_wavelength_matrix*), has in fact repeated wavelengths, which exist in the same link, but in different fibers (as two same wavelength signals cannot exist in the same fiber).

So, for example, if the list of wavelengths in a given link is [1,2,3,51,52,101] and *max_wavelengths* is 50, then the wavelengths 1, 51 and 101 are in fact the same wavelength, but in different fibers. The same is true for wavelengths 2 and 52. In the normalization step, the wavelengths are all mapped to a value between 1 and *max_wavelengths*. This is achieved through the modulo operation (the remainder of a division) between that wavelength's number and *max_wavelengths*. So in this example the *normalized_wavelengths* list would be [1,2,3,1,2,1]. Next, the maximum number of repetitions of wavelengths is determined based on the *normalized_wavelengths* list. In this case, wavelength 1 occurs 3 times, wavelength 2 occurs twice, and wavelength 3 only once. The maximum number of repeated wavelengths is thus 3, and so the number of fibers assigned to that link will be 3, as that is the number of fibers needed to accommodate all the wavelengths. If the list of wavelengths in a link is instead [1,52,103], a single fiber would be sufficient (the *normalized_wavelengths* list would be [1,2,3] and so the maximum number of repeated wavelengths would be 1).

This strategy guarantees that each “repeated” wavelength has its own fiber, ensuring that there are enough fibers to accommodate all the wavelengths, meaning that there are no optical channels with the same wavelength on the same fiber.

As previously mentioned, this strategy does not guarantee that the number of fibers is minimized when considering a network in its entirety. This is because, in certain situations, simply adding a fiber to a specific link can resolve blockages that occur in other links, as the traffic that would have been blocked in those links can be re-routed to the new alternative path, thus avoiding the addition of unnecessary fibers. However, an algorithm that exploits this type of solution could not be applied in the context of this work due to the high computational complexity inherent to this type of solution.

Nevertheless, in an attempt to improve the efficiency of the fiber assignment process and better optimize the addition of fibers, an alternative fiber assignment solution was also developed. This strategy does not consider the possibility of re-routing, as an ideal strategy would, but it tries to offer a middle-ground solution in terms of efficiency.

In this alternative strategy, the fiber assignment is made together with the wavelength assignment during the routing process (and not afterwards, as in the previous strategy). This results in a different approach, where instead of counting the number of fibers that need to be attributed given the wavelengths present in each link, the fibers are added during the routing and the wavelengths are attributed so as to minimize the addition of new fibers. The description of this solution is made in Appendix A.5.

While this method was shown to improve on the results of Algorithm 3, reducing the number of fibers introduced, the computation times ended up being very high, taking, for example, almost 28 hours to compute the routing solution in a single randomly generated network with 60 nodes considering a full-mesh logical topology. Computation times such as this make this solution inviable in the context of this thesis. For that reason, the method in Algorithm 3 was chosen as the one to use, as it ends up providing the best balance between efficiency in number of fibers added and fast computation.

4.3 Application of the Routing Algorithms

This section presents the results derived from the application of the routing algorithms previously described to both reference networks and randomly generated networks. The aim of this analysis is to understand the outputs of the RWA solution in different scenarios, establishing also a relationship between the different network’s parameters and its capacity.

4.3.1 Routing on Reference Networks

In this subsection, the constrained routing algorithm is applied to several reference networks: the DTAG/T-Systems National Core, the COST239, the UBN, the NSFNET and CESNET. The primary ob-

jective of this analysis is to present the network parameters for each of these reference networks and understand the impact of the link capacity limitations on the blocking of traffic demands. Additionally, this analysis seeks to comprehend how different sorting strategies - shortest-first, longest-first, and largest-first - influence the routing solution. The topological representations of these networks can be found in Appendix B.1.

The parameters related to the physical topology of the reference networks, the same parameters considered as inputs to the DNN models, are presented in Table 4.1. These parameters are determined through *routing_nx* and are independent of the applied routing strategy.

Table 4.1: Physical Topology Parameters for Five Reference Networks

	DTAG	**COST239**	**UBN**	**NSFNET**	**CESNET**
Number of Nodes	14	11	24	14	7
Number of Links	23	26	43	21	9
Min. Link Length (km)	80	171	250	246	173.75
Max. Link Length (km)	480	953	2600	2828	425.25
Avg. Link Length (km)	236.52	462.62	993.23	1080.62	296.84
Var. of Link Length (km^2)	13622.68	42522.39	133074.69	485574.9	5980.67
Minimum Node Degree	2	4	2	2	2
Maximum Node Degree	6	6	5	4	3
Average Node Degree	3.29	4.72	3.58	3	2.57
Variance of Node Degree	1.204	0.3801	0.909	0.2857	0.24
Network Diameter	6	4	8	5	4
Algebraic Connectivity	88.6	943.1	290.47	732.66	268.85

Table 4.1 provides a detailed comparison of the characteristics of each network, highlighting the diverse features inherent to these networks. A common feature across all networks is the minimum node degree of at least 2, which is consistent with the conclusions of [9], underscoring the robustness of these networks. Note that the units km^2 in the link length variance stem from Eq. (2.3).

In order to examine the impact of the link capacity and the influence of various sorting orders on the routing solutions, constrained routing was applied to three reference networks: CESNET, NSFNET, and UBN. These networks were chosen so as to represent differently sized networks. In these tests the routing was done considering a full-mesh logical topology (described by Eq. (4.1)). The sorting

strategies shortest-first and longest-first (described in Section 2.1.3) are done according to path length. The largest-first strategy is done according to the traffic value (defined in the traffic matrix), but since in this case the traffic matrix's values are all unitary (Eq. (4.1)), this sorting order ends up following the node numbering (routing first the demands to and from node 1, then to and from node 2, etc.). The link capacities were set to 75 (corresponding to 64 Gbaud transmission in the C-band, as explained in Section 2.2.1), and to a stricter limit (depending on the network) meant to purposefully cause blocking. The results for the routing analysis on the CESNET, NSFNET and UBN are presented in Table 4.2 to Table 4.4. Note that the blocking probability presented in the tables is determined as the division of the number of blocked traffic demands by $N(N-1)$. The 'Sho', 'Lon' and 'Lar' columns refer to the shortest-first, longest-first and largest-first sorting strategies, respectively.

Table 4.2: Capacitated Routing results on the CESNET

CESNET	**Link Capacity = 75**			**Link Capacity = 3**		
	Sho	**Lon**	**Lar**	**Sho**	**Lon**	**Lar**
Blocked Traffic Demands	0	0	0	12	18	16
Blocking Probability	0	0	0	0.286	0.429	0.38
Min. Path Length (km)	173.75	173.75	173.75	173.75	173.75	173.75
Max. Path Length (km)	1043.66	1043.66	1043.66	1350.94	1043.66	817.59
Avg. Path Length (km)	525.18	525.18	525.18	451.87	553.19	443.74
Avg. Hops per Path	1.76	1.76	1.76	1.53	1.83	1.46
Avg. Load per Link	4.11	4.11	4.11	2.55	2.44	2.11

Table 4.3: Capacitated Routing results on the NSFNET

NSFNET	**Link Capacity = 75**			**Link Capacity = 15**		
	Sho	**Lon**	**Lar**	**Sho**	**Lon**	**Lar**
Blocked Traffic Demands	0	0	0	4	14	6
Blocking Probability	0	0	0	0.021	0.077	0.033
Min. Path Length (km)	246	246	246	246	246	246
Max. Path Length (km)	4444	4444	4444	5973	9508	8801
Avg. Path Length (km)	2268.32	2268.32	2268.32	2358.9	2900.7	2849
Avg. Hops per Path	2.41	2.41	2.41	2.42	2.71	2.55
Avg. Load per Link	10.47	10.47	10.47	10.24	10.86	10.67

Table 4.4: Capacitated Routing results on the UBN

UBN	**Link Capacity = 75**			**Link Capacity = 30**		
	Sho	**Lon**	**Lar**	**Sho**	**Lon**	**Lar**
Blocked Traffic Demands	0	0	0	40	68	40
Blocking Probability	0	0	0	0.072	0.123	0.072
Min. Path Length (km)	250	250	250	250	250	250
Max. Path Length (km)	6959	6959	6959	9650	6959	7700
Avg. Path Length (km)	2995.13	2995.13	2995.13	2890.6	3407.3	3187.1
Avg. Hops per Path	3.17	3.17	3.17	3.02	3.51	3.25
Avg. Load per Link	20.35	20.35	20.35	17.98	19.77	19.33

Several observations can be made from the results above: (1) when considering a link capacity of 75 there is no blocking in any of the networks. This indicates that, for networks of this size (with this number of nodes) and pattern of traffic, transmission on the C-band and through links with a single fiber pair (one fiber in each communication direction) is sufficient and there is no need to consider an SDM solution; (2) when there is no blocking (in the case of link capacity equal to 75), the results are equal for every sorting strategy (shortest-first, longest-first, and largest-first). This uniformity in results occurs because, at the end of the process, since every traffic demand is routed, the results end up being the same, independently of the order used to route them; and (3) when blocking occurs (i.e., when the link capacity is reduced), the shortest-first strategy consistently performs the best in all three networks. This suggests that the shortest-first strategy is the most efficient in terms of minimizing blocking when the network capacity is constrained.

Relative to this routing analysis, bar charts with the number of optical channels in each link (link loads) were obtained for the three reference networks. These bar charts are presented in Appendix B.2.

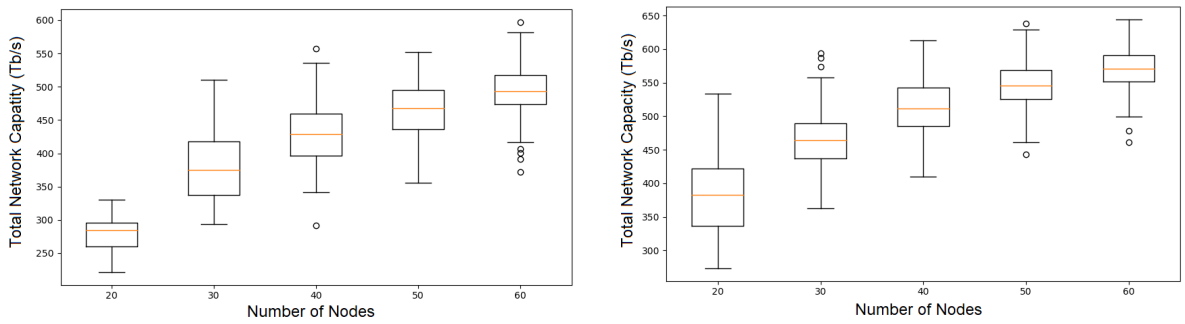
4.3.2 Network Capacity According to the Number of Nodes and Symbol Rate

In this subsection, the total network capacity is analyzed according to the network size in number of nodes. Transmission at 64 Gbaud and 128 Gbaud is compared, as well as the results of the two routing strategies. Throughout this analysis the routing was done considering a full-mesh logical topology, as described by Eq. (4.1), with the shortest-first sorting order. The maximum number of optical channels per fiber was set at 75 for 64 Gbaud transmission, and 37 for 128 Gbaud transmission. These values result from the fixed bandwidth of 4800 GHz in C-band transmission and by considering a channel spacing of 64 GHz and 128 GHz for 64 Gbaud and 128 Gbaud transmission, respectively, as explained in Section 2.2.1. The capacities of the optical channels were determined based on the optical reach in Table 2.1, using the respective values for 64 Gbaud and 128 Gbaud transmission accordingly. The results presented in this subsection were published on the article [4].

This analysis was conducted on five sets of networks, each set containing 200 random networks. These networks were generated using the *random_network* program, detailed in Chapter 3. The networks, varying in node count from 20 to 60 in increments of 10, were generated within a plane of 1000 km side length, using Waxman parameters $\alpha = \beta = 0.4$, and with an average node degree varying between 2 and 4.

Constrained Routing Scenario

Figure 4.1 represents the box-plots of the total network capacity according to the number of nodes determined for a constrained routing scenario. Each box-plot refers to a given set with 200 networks.

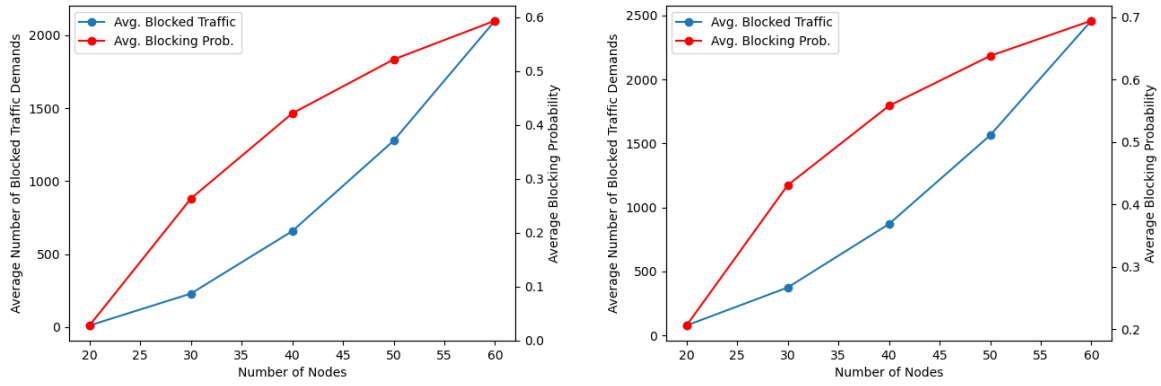


(a) Transmission at 64 GBaud **(b)** Transmission at 128 GBaud

Figure 4.1: Total Network Capacity (Tb/s) according to the number of nodes.

From Fig. 4.1, it can be inferred that as the number of nodes increases, so does the network capacity, albeit at a decreasing rate for higher node counts. This trend is in line with the relationship between the number of nodes and the network capacity as described in Eq. (2.8). The slower growth in network capacity with an increase in the number of nodes suggests an increase in the blocking of traffic demands.

As illustrated in Fig. 4.2, both the number of blocked traffic demands and the blocking probability (calculated by dividing the number of blocked traffic demands by $N(N-1)$) confirm this trend. The rise in blocking is due to the fact that networks with a larger number of nodes handle a greater amount of traffic demands, given the full-mesh logical topology under consideration (described by Eq. (4.1)). As a result, the conditions that lead to blocking (as detailed in Section 4.1) become more prevalent as the links become more saturated with traffic demands.



(a) Transmission at 64 GBaud **(b)** Transmission at 128 GBaud

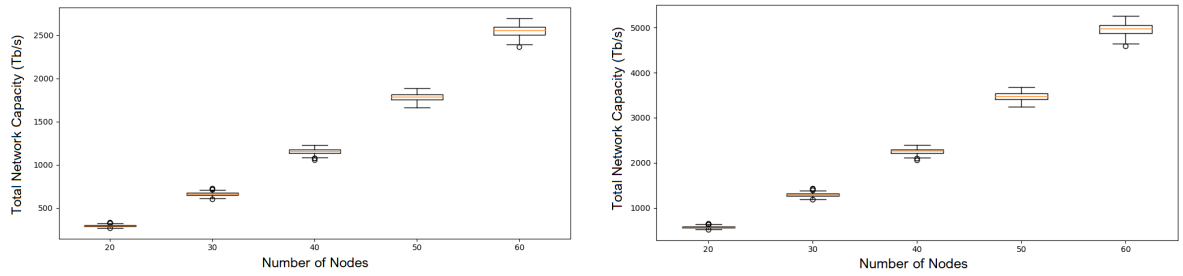
Figure 4.2: Average number of blocked traffic demands and average blocking probability as a function of the number of nodes (average across each set).

When examining the transmission at 64 Gbaud (Fig. 4.1(a)) and at 128 Gbaud (Fig. 4.1(b)), it's evident that a symbol rate of 128 Gbaud allows for a greater total network capacity compared to the 64 Gbaud scenario. When going from 64 Gbaud to 128 Gbaud, the enhancement in network capacity across all sets of networks is on average 22%. However, the degree of this enhancement tends to diminish as the number of nodes increases. Comparing the median capacity values across the sets, the transition from 64 Gbaud to 128 Gbaud transmission results in an approximate improvement of: 34%, 24%, 19%, 17%, and 16%, for the respective sets of graphs, ordered by increasing number of nodes. The observed reduction in performance improvement can be attributed to an increase in the blocking probability in the larger networks. For instance, for a network of 60 nodes, the blocking probability increases from 0.6 to 0.7 when the symbol rate goes from 64 Gbaud to 128 Gbaud, as illustrated in Fig. 4.2. Nevertheless, the increase in network capacity, along with the decrease in the number of wavelengths, underscores the significant benefit of using 128 Gbaud over 64 Gbaud.

Unconstrained Routing with Fiber Assignment Scenario

To address the blocking problem, the SDM solution, described in Section 4.2, can be implemented. In this scenario, since the blocking is eliminated by the addition of fibers, the total length of deployed fiber becomes a relevant metric.

Figure 4.3 represents the box-plots for the unconstrained network capacity determined in this scenario.



(a) Transmission at 64 GBaud

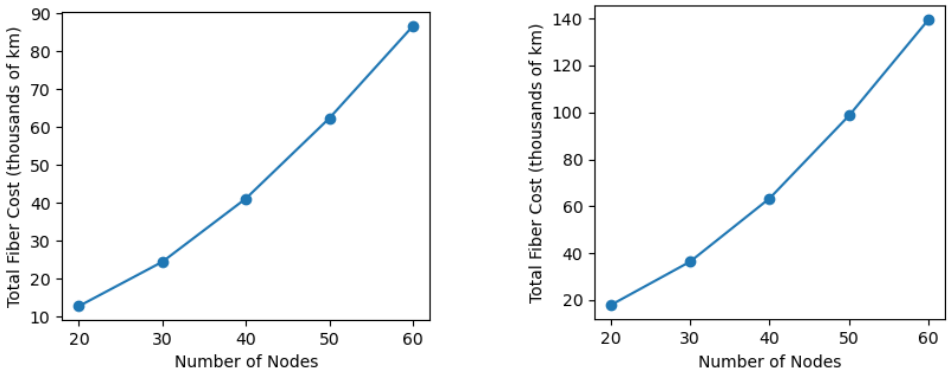
(b) Transmission at 128 GBaud

Figure 4.3: Unconstrained Total Network Capacity (Tb/s) according to the number of nodes.

From the analysis of Fig. 4.3, it is noticeable that the total network capacity tends to exhibit a quadratic growth pattern with respect to the number of nodes ($\sim N^2$). This scenario also demonstrates the achievement of substantial capacities, with networks with 60 nodes reaching approximately 2.5 Pbit/s at a symbol rate of 64 Gbaud, as depicted in Fig. 4.3(a). Furthermore, the median values of the total network capacity for a 30 node network, approximately 660 Tbit/s, align with the values delineated in Fig. 9 of [38] for a 30 node CONUS topology, which was generated using the Erdós-Rényi model. As

anticipated, a twofold increase in the total capacity is observed when the symbol rate is augmented to 128 Gbaud, as illustrated in Fig. 4.3(b).

However, this significant enhancement in capacity is accompanied by a considerable increase in the network cost, manifested as an escalation in the length of the optical fiber required for deployment. This correlation is demonstrated in Fig. 4.4, which presents the total fiber cost (quantified in terms of the total fiber length in thousands of kilometers) as a function of the number of nodes.



(a) Transmission at 64 GBaud **(b)** Transmission at 128 GBaud

Figure 4.4: Average Fiber Cost in thousands of km (average across each set).

This figure presents a relationship between the fiber cost and the number of nodes which, as with the unconstrained capacity, shows an approximately quadratic variation. A key insight from Fig. 4.4 is that an increase in the symbol rate from 64 Gbaud to 128 Gbaud results in an approximate 51% increase in the total fiber cost. However, this increase in cost is accompanied by a doubling of the total network capacity, as previously observed.

4.3.3 Capacity According to Average Link Length

To understand how the generated networks' capacity and average channel capacity varies relative to the network's physical size, an analysis on the average link length was made. With that purpose, a set of 3000 networks with 20 nodes was generated considering $\alpha = \beta = 0.4$, and a square plane varying from 1000 km to 10000 km in side length (500 networks generated in a plane with side length of 1000 km, 500 in a plane with 2000 km length, 500 in a plane with 3000 km length, 500 in a plane with 5000 km length, 500 in a plane with 7000 km length, and 500 networks in a plane with 10000 km length). This plane size variation as well as the lower number of nodes were considered so as to assure a larger variability in average link length. The constrained routing was done with the shortest-first sorting strategy and with a limit of 75 optical channels per link considering also a full-mesh logical topology.

Figure 4.5 depicts two scatter plots with the average channel capacity and the total network capacity according to the average link length. Each point in the scatter plot represents a single network, indicating the relation between that network's capacity and average link length. It can be seen that both the average channel capacity and the total network capacity tend to decrease (approximately) logarithmically with the average link length. This behavior is consistent with what was concluded in [39], where it was shown that both of these capacities decrease logarithmically with the increase of average link length.

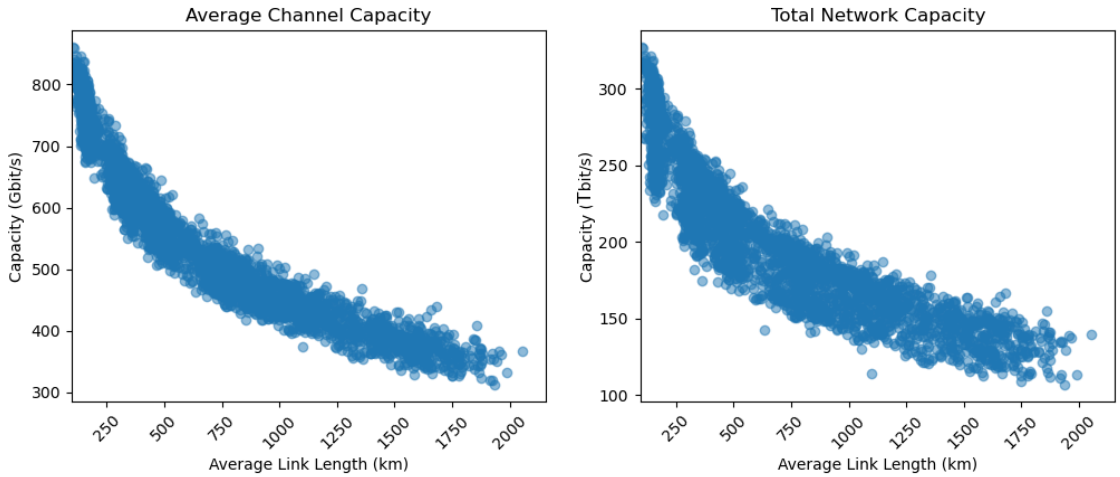


Figure 4.5: Average channel capacity (left) and total network capacity (right) according to the average link length.

4.4 Chapter Conclusions

This chapter details the implementation of the routing algorithms and their application to both reference and randomly generated networks. The development of these two routing solutions led to the creation of a routing heuristic, as well as fiber assignment heuristics that permit a complete analysis of the networks, allowing for an efficient determination of their parameters and capacities, to be used for the training and testing of the DNN models.

It was shown that, for constrained routing with a full-mesh logical topology and a limit of 75 optical channels per link (equivalent to 64 Gbaud transmission in the C-band), there is typically sufficient link capacity to prevent traffic demand blocking in smaller networks. However, as the networks increase in number of nodes (and traffic demands increase as well), blocking begins to grow, becoming a key factor limiting network capacity growth. When the link capacity limitation is overcome through the solution of routing with fiber assignment, the unconstrained network capacity can reach significant values, exhibiting an approximately quadratic growth. On the other hand, the cost in kilometers of deployed fiber also tends to grow in a similar way.

Chapter 5

Implementation of the DNN Models

In this chapter, the training and testing procedures for two developed DNN regression models are discussed. The first model aims to predict the total network capacity and the average channel capacity subject to the restriction imposed by the limited number of optical channels in the network's links. The second model aims to predict the total network capacity and the total cost in kilometers of fiber for the case where that limitation is overcome through the addition of optical fibers to the network (SDM solution). Both models consider the same inputs, which are parameters related to a network's physical topology (the 12 parameters listed in Section 4.1.1). The main objective of using the DNN models is to obtain the capacity values in a quick and reliable way, avoiding the longer computation times inherent to the routing process. For that reason, throughout this chapter the computation times from the routing algorithms and the DNN models are compared alongside the results and predictions.

5.1 Training the DNN Models

The training of the DNN models, as detailed in Section 2.3.3, requires the use of a large set of features (inputs) and labels (outputs) from which the model's parameters can be determined. This data was obtained from the application of the routing strategies described in Chapter 4, to sets of random networks generated with the model described in Chapter 3. Therefore, the software tools developed throughout this dissertation project are all used for the development of the DNN models: the program *random_network* is used to generate large sets of random networks and then *routing_nx* (or the alternative version which implements the SDM solution, *routing_nx_no_blocking*) is used to determine the network's physical topology parameters (corresponding to the features) as well as the capacities and fiber cost (the labels) determined from the routing solution.

5.1.1 First DNN Model: Total Network Capacity and Average Channel Capacity in a Constrained Routing Scenario

A set with 15245 networks was used to train the first DNN model. These networks were generated considering 2D planes with side lengths varying from 1000 km to 5000 km in increments of 1000 km, number of regions in the plane set to 4, number of nodes varying from 5 to 55, and an average node degree varying from 2 to around 5. The Waxman parameters chosen were $\alpha = \beta = 0.4$. This range of number of nodes was chosen as it covers the values typically found in reference networks [9] while also allowing for a short computation time per network with the *routing_nx* tool. This allows for the use of a larger dataset (with more examples), which can be beneficial to the training process.

The determination of the constrained routing solution was done considering a full-mesh logical topology, the shortest-first sorting strategy, and transmission at 64 Gbaud in the C-band, resulting in a limit of 75 optical channels per link. The computation of this solution with *routing_nx* took around 3 hours 42 minutes for the entire set of networks. Table 5.1 shows the statistical analysis of the data obtained.

Table 5.1: Statistics of the features and labels for the first DNN model

Parameter	Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum
Number of Nodes	5	22	36	33.787	46	55
Number of Links	5	37	59	60.330	82	138
Minimum Link Length (km)	75	79	90	109.794	112	692
Maximum Link Length (km)	204	880	1497	1604.580	2191	5042
Average Link Length (km)	119.542	326.459	534.583	550.304	748.869	2260
Variance of Link Length (km^2)	968	34685.73	100575.99	141063.71	208868.31	1953209
Minimum Node Degree	2	2	2	2.039	2	5
Maximum Node Degree	2	5	6	6.364	8	14
Average Node Degree	2	2.824	3.579	3.574	4.308	5.143
Variance of Node Degree	0	0.667	1.391	1.582	2.321	6.678
Diameter	1	7	9	9.325	11	27
Algebraic Connectivity	1.933	38.889	99.140	217.339	222.909	8351.290
Total Network Capacity (Tb/s)	11.8	254.077	446.96	514.873	733.899	1934.960
Avg. Channel Capacity (Gb/s)	211.919	439.517	533.001	538.792	632.738	970

Table 5.1 shows that there is a great variability in the network parameters. This is meant to reflect

the variability in real-world optical backbone networks, training the model for a wide range of possible networks.

Data Pre-Processing

As explained in Section 2.3.3, an important initial phase in the training of neural networks is the pre-processing of the data. As such, the correction of skewness and the standardization of the features were performed.

Skewness is a measure of the asymmetry in the statistical distribution of the data. The data can be right-skewed (or with positive skewness) when the tail of the distribution extends towards higher values, on the right side of the peak, or it can be left-skewed (with negative skewness) when the tail of the distribution stretches towards lower values, on the left side of the peak [40].

To address this, either the square root or the square function are applied to the features, depending on the direction of the skew. The square root function $(\sqrt{x})$ is used for right-skewed data and the square function (x^2) is used for left-skewed data [40].

By individually visualizing the distribution of each feature, these transformations can be applied accordingly, if deemed necessary. The fixing of the features' skew is essential for the performance of the DNN models, as it ensures that the models are not biased towards particular ranges of feature values, while also reducing the impact of potential outliers [40].

Following the correction of skewness, standardization was performed. Standardization is a common pre-processing step in machine learning that involves rescaling the features to have a mean of 0 and a standard deviation of 1 [30]. This is done to ensure that all features are on the same scale, contributing equally to the model, regardless of their original scale. The standardization of a feature (x) is calculated as follows:

$$z = \frac{x - \mu}{\sigma}, \tag{5.1}$$

where x is the original feature vector, μ is the mean of the feature vector, and σ is its standard deviation. This process was applied to all features in the dataset prior to training the DNN models.

After the correction of skewness and standardization, the next step was the shuffling and splitting of the dataset. Shuffling the data serves to remove any inherent order in the data that might bias the training of the DNN models. The shuffled data was then split into three subsets: a training set, a validation set, and a test set. The training set, which comprised 70% of the total data, was used to train the DNN model. The validation set, comprising 15% of the total data, was used to fine-tune the model's hyperparameters and prevent overfitting. Finally, the test set, also comprising 15% of the total data, was used to evaluate the performance of the trained models on unseen data.

Hyperparameter Tuning

In the training of the DNN model some configurations were predefined before the training process itself, while others, the hyperparameters, were set to be tuned during the training process.

The predefined configurations were: (a) the activation function, set as the ReLU for all layers except the output layer, where a linear activation function was used, as is typical in regression problems; (b) the iterative optimization method, set as the SGD, due to its simplicity and fewer parameters that might need tuning when compared with the Adam; (c) the loss function, set as the MSE, a common choice in regression problems; and (d) the maximum number of epochs, set to 2000, with early stopping occurring after 20 epochs without validation loss improvement (patience parameter of 20).

The hyperparameters are: (i) the number of hidden layers; (ii) the number of hidden units (defining the number of neurons in each hidden layer, the same number on every hidden layer); (iii) the dropout probability, relative to the dropout regularization method; (iv) the learning rate; and (v) the batch size.

The hyperparameter tuning process is conducted by evaluating the model's performance on the validation set for each unique combination of hyperparameter values. The combination that yields the best performance is then selected. The strategy employed for this hyperparameter tuning process is the Grid Search method [41]. In this method, each hyperparameter is assigned a predefined list of discrete values and every possible combination of these values is evaluated. This process was conducted in two stages. In the first stage, the number of hidden layers, the number of hidden units, and the learning rate are optimized. Following the selection of the best-performing hyperparameters from the first stage, the second stage involves tuning the batch size and the dropout probability. The specific values tested for each hyperparameter, also known as the search space, are presented in Table 5.2.

Table 5.2: Hyperparameter Search Space

Hyperparameter	Search Space
Number of Hidden Layers	0, 1, 2, 3
Number of Hidden Units	10, 50, 100
Learning Rate	0.1, 0.01, 0.001
Batch Size	32, 64, 128
Dropout Probability	0, 0.1, 0.2

The search spaces in Table 5.2 were chosen to ensure a sufficiently diverse exploration of model configurations. During the first stage of hyperparameter tuning, when the first three hyperparameters are tested, the batch size was set to 64 and the dropout probability was set to 0.

Table 5.3 shows the results of the first stage of hyperparameter tuning, presenting the performance of the DNN model on the validation set measured with the R^2 score (Eq. (2.21)) and the ARE (Eq. (2.22)) for each combination of the hyperparameters' search space values. The table also includes the number of model parameters (Eq. (2.24)). In this table $y1$ refers to the output total network capacity and $y2$ to the average channel capacity.

Table 5.3: Results on the validation set of the first stage of hyperparameter optimization for the first DNN model. y1: Total Network Capacity; y2: Avg. Channel Capacity

Hidden Layers	Hidden Units	Learning Rate (η)	R^2 score $y1$	R^2 score $y2$	ARE $y1$	ARE $y2$	DNN Parameters
0	-	0.1	0.9556	0.9560	0.1302	0.0461	26
0	-	0.01	0.9564	0.9562	0.1300	0.0455	26
0	-	0.001	0.9533	0.9559	0.1284	0.0458	26
1	10	0.1	0.9754	0.9774	0.0682	0.0308	152
1	10	0.01	0.9749	0.9776	0.0677	0.0310	152
1	10	0.001	0.9716	0.9746	0.0782	0.0335	152
1	50	0.1	0.9783	0.9801	0.0622	0.0292	752
1	50	0.01	0.9768	0.9795	0.0643	0.0295	752
1	50	0.001	0.9752	0.9781	0.0692	0.0303	752
1	100	0.1	0.9763	0.9789	0.0657	0.0297	1502
1	100	0.01	0.9777	0.9797	0.0654	0.0293	1502
1	100	0.001	0.9765	0.9786	0.0685	0.0302	1502
2	10	0.1	0.9766	0.9784	0.0647	0.0305	262
2	10	0.01	0.9773	0.9792	0.0598	0.0294	262
2	10	0.001	0.9741	0.9770	0.0685	0.0308	262
2	50	0.1	0.9777	0.9803	0.0641	0.0294	3302
2	50	0.01	0.9782	0.9802	0.0609	0.0290	3302
2	50	0.001	0.9769	0.9790	0.0659	0.0298	3302
2	100	0.1	0.9795	0.9813	0.0590	0.0282	11602
2	100	0.01	0.9785	0.9803	0.0614	0.0288	11602
2	100	0.001	0.9776	0.9796	0.0644	0.0294	11602
3	10	0.1	0.9753	0.9780	0.0801	0.0311	372
3	10	0.01	0.9755	0.9777	0.0703	0.0307	372
3	10	0.001	0.9731	0.9760	0.0744	0.0325	372
3	50	0.1	0.9785	0.9805	0.0599	0.0289	5852
3	50	0.01	0.9782	0.9801	0.0616	0.0290	5852
3	50	0.001	0.9770	0.9791	0.0654	0.0297	5852
3	100	0.1	0.9785	0.9810	0.0622	0.0284	21702
3	100	0.01	0.9788	0.9806	0.0606	0.0287	21702
3	100	0.001	0.9779	0.9800	0.0635	0.0291	21702

As seen in Table 5.3, apart from the models with no hidden layers (linear models), most of the considered models tend to provide good performances, with R^2 scores in the range of 0.96 to 0.98 and average relative errors below 7%. Considering the goal of choosing the model with the best performance while also preferring fewer parameters, the model with 2 hidden layers with 10 hidden units each and the learning rate of 0.01 was selected. This model has relatively high R^2 scores (0.9773 for $y1$ and 0.9792 for $y2$) and low average relative errors (0.0598 for $y1$ and 0.0294 for $y2$). Additionally, it has a relatively low number of trained parameters (262), which represents a good balance between model complexity and performance. Figure 5.1 shows the training and validation loss against the number of epochs, depicting the model's smooth convergence to the minimum of the loss function in 990 epochs.

Table 5.4 shows the results of the second stage of the hyperparameter optimization, now considering

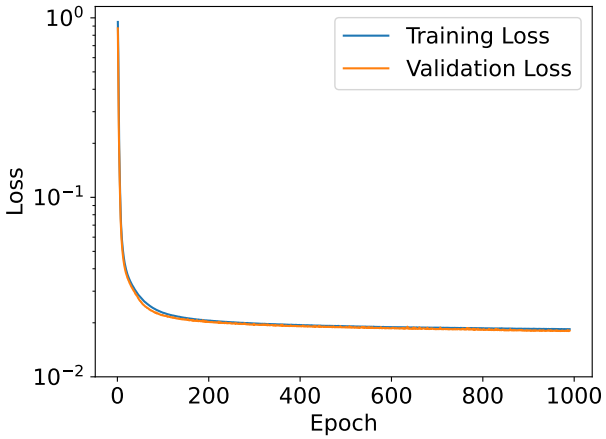


Figure 5.1: Training and Validation Loss (logarithmic scale) against number of epochs for the first DNN model.

the previously chosen parameters (2 hidden layers, 10 hidden units and learning rate of 0.01).

Table 5.4: Results on the validation set of the second stage of hyperparameter optimization for the first DNN model. y1: Total Network Capacity; y2: Avg. Channel Capacity

Batch Size	Dropout Prob.	R^2 score $y1$	R^2 score $y2$	ARE $y1$	ARE $y2$
32	0	0.9770	0.9788	0.0616	0.0297
32	0.1	0.9547	0.9568	0.1874	0.0461
32	0.2	0.9083	0.9112	0.3311	0.0673
64	0	0.9773	0.9792	0.0598	0.0294
64	0.1	0.9522	0.9537	0.1855	0.0479
64	0.2	0.9027	0.9073	0.3426	0.0684
128	0	0.9764	0.9787	0.0606	0.0299
128	0.1	0.9539	0.9536	0.1886	0.0477
128	0.2	0.9001	0.9074	0.3013	0.0663

From the results in Table 5.4, it can be observed that the model performs worse with higher dropout probabilities, especially when it comes to the total network capacity, as is clear when comparing the average relative errors, which increase significantly as the dropout probability increases. This indicates that the model may be underfitting the data when dropout is applied, meaning that the model is losing too much information during training, possibly due to the increased sparsity induced by dropout. Regarding the batch size, it can be seen that it does not affect the model's performance as much as the dropout does, which suggests that the model is relatively robust to changes in batch size within the range considered. Considering the best results in the table, the batch size of 64 was chosen.

The chosen hyperparameters for the first DNN model are then: 2 hidden layers, 10 hidden units, a learning rate of 0.01, a batch size of 64 and no dropout regularization. Figure B.12, in Appendix B.3, shows the structure of the DNN model chosen.

The results on the test set were the following: R^2 score for $y1$ was 0.9763 and for $y2$ it was 0.9785. The AREs were 0.0618 for $y1$ and 0.0301 for $y2$. Additional test to this model are presented in Section 5.2.

5.1.2 Second DNN Model: Total Network Capacity and Total Fiber Cost in a Scenario of Unconstrained Routing with Fiber Assignment

To train the second DNN model, a set with 8480 networks was generated with the *random_network* program. This set of random networks was generated under the same conditions as the set used for the first DNN model, but now with the number of nodes varying from 5 to 100. Since this model is used to predict the unconstrained network capacity and the total cost in kilometers of fiber in the case where the SDM solution is applied, the computation of the network parameters and the routing solution was done using *routing_nx_no_blocking*. As in the previous scenario, a logical full-mesh topology was considered, the shortest-first sorting strategy was used, and transmission was done at 64 Gbaud on the C-band (and so a limit of 75 wavelengths per fiber was set).

The larger range in number of nodes allows for this model to be used in a wider range of networks. However, the trade-off is that the number of examples used to train the model must be reduced due to the increased computational time associated to the routing process in larger networks. The computation time of *routing_nx_no_blocking* in the set of 8480 networks was 7 hours 51 minutes. Table 5.5 summarizes the characteristics of the obtained data.

Table 5.5: Statistics of the features and labels for the second DNN Model

Parameter	Minimum	1st Quartile	Median	Mean	3rd Quartile	Maximum
Number of Nodes	5	29	53	53.163	77	100
Number of Links	5	49	86	91.470	128	231
Minimum Link Length (km)	75	77	82	100.375	100	710
Maximum Link Length (km)	204	850.750	1550.500	1667.304	2316	6022
Average Link Length (km)	110.429	294.943	484.328	505.646	686.716	2591.722
Variance of Link Length (km^2)	1063.389	27867.126	91157.512	134928.28	203413.2	1841013
Minimum Node Degree	2	2	2	2.025	2	5
Maximum Node Degree	2	5	6	6.423	8	14
Average Node Degree	2	2.667	3.524	3.470	4.205	5.143
Variance of Node Degree	0	0.614	1.355	1.493	2.247	6.188
Diameter	1	8	11	12.477	15	45
Algebraic Connectivity	0.535	20.448	62.505	162.833	156.658	7536.313
Total Network Capacity (Tb/s)	12	492.550	1551.400	1976.071	3180.850	7277.800
Total Fiber Cost (km)	2044	63901	177013	265338.9	385248.5	1452770

As in the previous model, the dataset was pre-processed, shuffled and split into training, validation and test sets (as described in Section 5.1.1). The training process followed the same strategy of the first model, with the hyperparameter tuning being done according to the Grid Search method in two stages, considering the search spaces in Table 5.2. The results of the first stage of hyperparameter tuning are presented in Table 5.6, where $y1$ is the total network capacity and $y2$ is the total fiber cost.

Table 5.6: Results on the validation set of the first stage of hyperparameter optimization for the second DNN model. y1: Total Network Capacity; y2: Total Fiber Cost

Hidden Layers	Hidden Units	Learning Rate (η)	R^2 score $y1$	R^2 score $y2$	ARE $y1$	ARE $y2$	DNN Parameters
0	-	0.1	0.9907	0.9487	0.1149	0.2495	26
0	-	0.01	0.9897	0.9420	0.1132	0.2404	26
0	-	0.001	0.9898	0.9385	0.1124	0.2401	26
1	10	0.1	0.9991	0.9948	0.0247	0.0572	152
1	10	0.01	0.9992	0.9929	0.0296	0.0752	152
1	10	0.001	0.9965	0.9845	0.0707	0.1138	152
1	50	0.1	0.9994	0.9959	0.0239	0.0509	752
1	50	0.01	0.9991	0.9948	0.0297	0.0629	752
1	50	0.001	0.9980	0.9901	0.0561	0.0899	752
1	100	0.1	0.9993	0.9956	0.0246	0.0549	1502
1	100	0.01	0.9991	0.9951	0.0304	0.0598	1502
1	100	0.001	0.9980	0.9916	0.0532	0.0785	1502
2	10	0.1	0.9993	0.9947	0.0221	0.0564	262
2	10	0.01	0.9993	0.9950	0.0215	0.0625	262
2	10	0.001	0.9985	0.9898	0.0443	0.0917	262
2	50	0.1	0.9993	0.9955	0.0239	0.0539	3302
2	50	0.01	0.9992	0.9960	0.0257	0.0538	3302
2	50	0.001	0.9982	0.9930	0.0490	0.0762	3302
2	100	0.1	0.9993	0.9959	0.0246	0.0508	11602
2	100	0.01	0.9993	0.9957	0.0275	0.0549	11602
2	100	0.001	0.9986	0.9939	0.0423	0.0701	11602
3	10	0.1	0.9991	0.9935	0.0260	0.0747	372
3	10	0.01	0.9992	0.9951	0.0249	0.0621	372
3	10	0.001	0.9978	0.9888	0.0502	0.0872	372
3	50	0.1	0.9993	0.9945	0.0226	0.0545	5852
3	50	0.01	0.9993	0.9957	0.0238	0.0488	5852
3	50	0.001	0.9981	0.9921	0.0454	0.0662	5852
3	100	0.1	0.9994	0.9959	0.0239	0.0508	21702
3	100	0.01	0.9993	0.9956	0.0282	0.0541	21702
3	100	0.001	0.9985	0.9938	0.0456	0.0694	21702

Based on the results presented on Table 5.6, the set of hyperparameters chosen was: 1 hidden layer, 50 hidden units and a learning rate of 0.1. This configuration gives high R^2 scores (0.9994 for $y1$ and 0.9959 for $y2$), low average relative errors (0.0239 for $y1$ and 0.0509 for $y2$), and a relatively low number of trained parameters (752). This provides a good balance between model complexity and performance.

In Figure 5.2 the training and validation loss is plotted against the number of epochs showing the

model's convergence to the minimum of the loss function in 534 epochs.

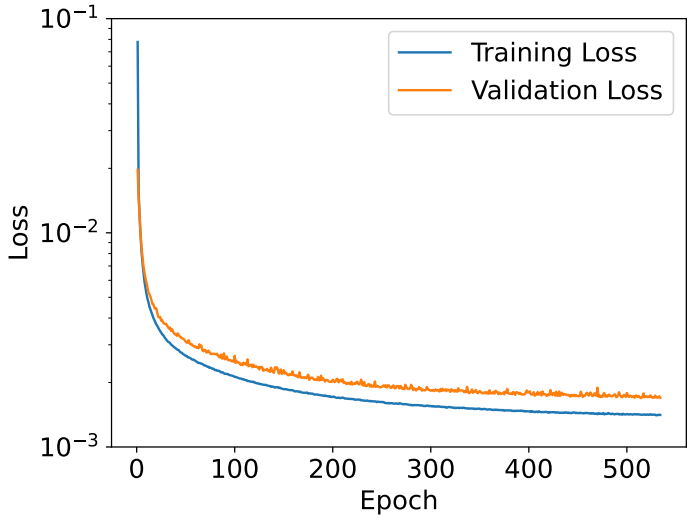


Figure 5.2: Training and Validation Loss (logarithmic scale) against number of epochs for the second DNN model.

In the second stage of the hyperparameter optimization, the batch size and the dropout probability are tested. Table 5.7 shows the results of that process.

Table 5.7: Results on the validation set of the second stage of hyperparameter optimization for the second DNN model. y1: Total Network Capacity; y2: Total Fiber Cost

Batch Size	Dropout Prob.	R^2 score $y1$	R^2 score $y2$	ARE $y1$	ARE $y2$
32	0	0.9994	0.9959	0.0245	0.0518
32	0.1	0.9985	0.9933	0.0588	0.0848
32	0.2	0.9982	0.9859	0.0628	0.0898
64	0	0.9994	0.9959	0.0239	0.0509
64	0.1	0.9989	0.9945	0.0346	0.0774
64	0.2	0.9985	0.9929	0.0453	0.0847
128	0	0.9992	0.9955	0.0269	0.0560
128	0.1	0.9988	0.9938	0.0394	0.0792
128	0.2	0.9982	0.9918	0.0762	0.0913

From the results presented in Table 5.7, similarly to the first model, we can see that the best performance is for a dropout probability of 0 (no dropout regularization applied) and for a batch size of 64, as these are the hyperparameters that result in lower ARE values.

So, the hyperparameters chosen for the second DNN model after the two stages of the optimization process are: 1 hidden layer, 50 hidden units, learning rate 0.1, batch size of 64 and no dropout regularization. The architecture of the chosen DNN model is represented in Figure B.13, in Appendix B.3.

The results on the test set were the following: R^2 score of 0.9994 for $y1$ and 0.9962 for $y2$; average relative error of 0.0243 for $y1$ and 0.0512 for $y2$. Additional tests were performed on this model, being presented in the next section.

5.2 Testing the DNN Models

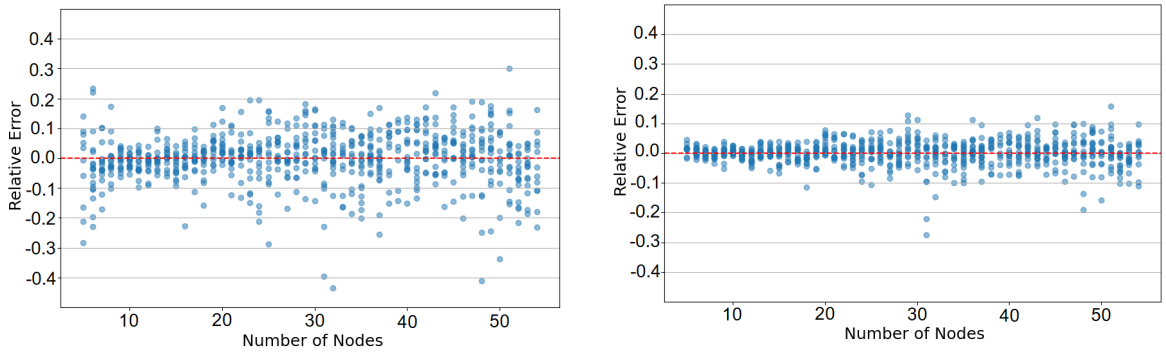
In order to evaluate the predictive performance of the DNN models, tests were conducted on both randomly generated networks and reference networks. The main objective of this analysis is to compare the predictions and prediction times of the DNN models with the outcomes and computation times of the routing solutions implemented using the *routing_nx* and *routing_nx_no_blocking* tools.

5.2.1 Tests on the First DNN Model

To evaluate the first DNN model, an additional test set with 750 random networks was generated. These networks were generated with the same parameters of node count, 2D plane length, and average node degree as the dataset used for training this model. The routing process was also conducted under the same conditions as those used for the training set. The total computation time with *routing_nx* on this set of networks was approximately 7 minutes and 10 seconds. In contrast, the DNN model was able to make the predictions for the entire set of 750 networks in just 22 milliseconds.

The average relative errors for this test set, calculated with Eq. (2.22), were: 6.17% for the total network capacity predictions and 2.84% for the average channel capacity predictions. While these average values provide an overview of the model's performance across the set, it is equally important to examine the error values for individual examples.

Figure 5.3 shows the scatter plot of the relative errors against the number of nodes for both outputs. Each dot represents the relative error (given by Eq. (2.23)) between the value determined from the routing solution (y_i) and the prediction made with the DNN model ($\hat{y}_i$) for an individual network.



(a) Total Network Capacity

(b) Average Channel Capacity

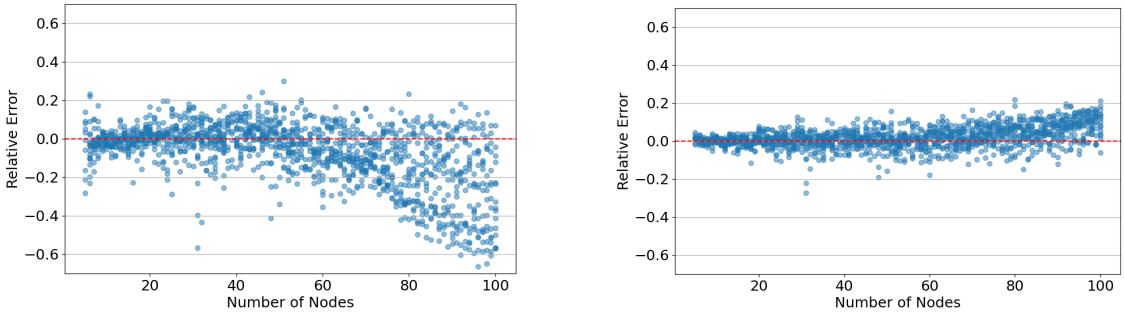
Figure 5.3: Relative error against number of nodes for both outputs of the first DNN model.

As depicted in Fig. 5.3(a), for the total network capacity output, the majority of relative errors are under 10%. Specifically, 79.23% of the examples exhibit a relative error below 10%, and 90.81% of exam-

ples have a relative error below 15%. The average channel capacity output, as illustrated in Fig. 5.3(b), generally presents even smaller relative errors: 83.36% of examples have a relative error below 5%, and a significant 97.60% of the examples have a relative error below 10%.

These results show that, despite providing a good performance in both outputs, this is not uniform across the two outputs, as the model exhibits a superior performance for the average channel capacity, having smaller relative errors (Fig. 5.3(b)). The observed disparity in performance between the two outputs could be indicative of the model's ability to learn and generalize from the training data, with the superior performance on the average channel capacity suggesting that the features present in the training data could be more predictive for this output, allowing the DNN model to learn more accurately.

To test the model's performance on larger networks, a test set with 1440 networks was generated in the same conditions as the previous sets, but now with a number of nodes varying from 5 to 100. The runtime of *routing_nx* for this set was 2 hours 53 minutes. The predictions with the DNN model took only 63 milliseconds for the entire set. The scatter plot of the relative errors against the number of nodes for this set of networks is shown on Figure 5.4.



(a) Total Network Capacity

(b) Average Channel Capacity

Figure 5.4: Relative error against number of nodes for both outputs of the first DNN model (N in [5,100]).

From the plots in Figure 5.4, it is easy to understand that the model's performance in networks with a larger number of nodes is significantly more irregular, becoming worse as the number of nodes grows. This behavior is expected as the model has been trained on a specific range of data (networks with a number of nodes from 5 to 55), and extrapolating beyond this range can lead to less reliable predictions. This is called Out-of-Distribution (OOD) generalization [42], and it is a problem that conventional supervised learning methods (such as DNNs) cannot effectively solve (in these types of models it is a fundamental assumption that the training and test datasets originate from the same distribution). Note also that, as the number of nodes increases, parameters such as the number of links and even the outputs will also tend to be OOD, further affecting the generalization capabilities of the model.

The inaccurate performance is particularly evident in the total network capacity (Fig. 5.4(a)). In this

case the predictions tend to stay inside the range of 20% relative error when the number of nodes is lower than 55, but for higher numbers of nodes the relative errors start progressively becoming higher (in absolute value). However, in the case of the average channel capacity (Fig. 5.4(b)), while there is still a degradation in the model's performance, this effect is much less pronounced than in the case of the total network capacity, with the relative errors staying inside the 20% range even for larger node values. This may indicate that the prediction of this output may be more dependent on other features that do not vary (much) outside their original distribution range or that the output itself does not vary (as much) to OOD values as the total network capacity does.

Nevertheless, it is important to note that, when dealing with OOD generalization, the model's behavior can be very unpredictable and erratic, and so an evaluation on the model's performance poses a significant challenge [42]. Addressing OOD generalization is a current topic of research in the field of ML [42].

Table 5.8 shows the results of the DNN model's predictions and the routing solution values for specific random networks with a varying number of nodes. The time needed to determine the capacity values with *routing_nx* is compared with the time taken for the DNN model to make the predictions.

Table 5.8: First model's predictions in random networks. y1: Total Network Capacity; y2: Avg. Channel Capacity

Number of Nodes	$y1$ (Tb/s)	Pred. $y1$ (Tb/s)	Relative Error $y1$ (%)	$y2$ (Gb/s)	Pred. $y2$ (Gb/s)	Relative Error $y2$ (%)	Routing time (s)	Pred. time (ms)
10	48.0	48.67	1.40	533.33	547.08	2.58	0.19	51.67
20	303.2	303.75	0.18	797.89	801.63	0.47	0.20	60.32
30	708.0	663.32	6.31	813.79	787.77	3.19	0.38	24.70
40	292.67	268.64	8.21	333.33	332.92	0.12	0.43	10.02
50	627.98	530.97	15.45	376.49	342.26	9.09	1.97	8.56
60	600.271	718.48	19.69	315.93	331.48	4.92	2.49	22.86
70	1080.11	1141.53	5.69	412.26	386.95	6.14	5.97	13.59
80	1293.15	1476.62	14.18	334.49	353.83	5.78	21.38	16.89
90	2204.88	1964.20	10.92	463.80	387.0	16.56	38.43	12.31
100	1641.51	1786.44	8.82	314.46	294.91	6.22	56.93	21.60

As a first conclusion, we can see that the difference in computation times between the *routing_nx* calculations and the DNN model's predictions is very evident, with the DNN model never taking more than a few tens of milliseconds, while *routing_nx* can take up to tens of seconds in larger networks. Furthermore, when it comes to the capacity values, the same trends identified before can be noticed here as well: the errors tend to be smaller for y_2 and for networks with a number of nodes inside the training range. Nevertheless, it is important to note that there is inherent variability in the data that can lead to exceptions to the overall trends. This is evidenced, for example, by the network with 100 nodes, which achieves better results than some networks with fewer nodes. This does not necessarily contradict the overall trend, but rather highlights the stochastic nature of the data and the model's predictions.

To assess the performance of the DNN model in real-world networks, five reference networks (whose topologies are represented in Appendix B.1) were considered. Table 5.9 compares the capacity values determined with *routing_nx* with the DNN model's predictions and the computation times.

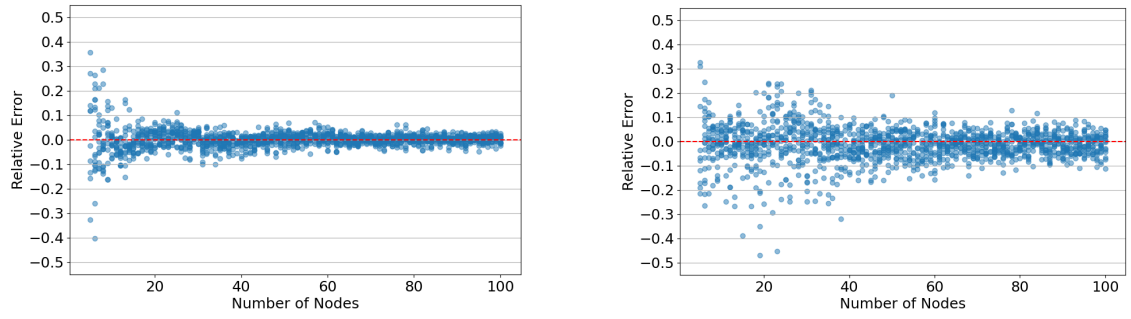
Table 5.9: First model's predictions in reference networks. y1: Total Network Capacity; y2: Avg. Channel Capacity

Network Name	$y1$ (Tb/s)	Pred. $y1$ (Tb/s)	Relative Error $y1$ (%)	$y2$ (Gb/s)	Pred. $y2$ (Gb/s)	Relative Error $y2$ (%)	Routing time (ms)	Pred. time (ms)
CESNET	32.6	35.75	9.65	776.19	792.29	2.07	260.55	20.84
COST239	81.2	80.05	1.42	738.18	730.69	1.02	224.98	10.01
DTAG	147.4	141.64	3.91	809.89	783.61	3.24	174.35	23.01
NSFNET	98.0	96.85	1.17	538.46	543.50	0.94	168.70	22.39
UBN	272.8	262.85	3.65	494.20	472.54	4.39	263.46	22.72

The results show that the DNN model predicts both outputs with low relative errors. This indicates that the model is able to make accurate predictions in real-world optical backbone networks, which further confirms that the topologies of the generated networks (used in the training of the model) are close to their real-world counterparts. It is worth noting, none the less, that the y_1 prediction for the CESNET shows a higher relative error compared to the other networks. This could be attributed to that network's specific characteristics which fall in a range where the model's performance is less consistent.

5.2.2 Tests on the Second DNN Model

To test the performance of the second DNN model, the same set of 1440 networks (with a number of nodes ranging from 5 to 100) used to test the previous model was considered. The unconstrained routing with fiber assignment was done with the *routing_nx_no_blocking* program under the same assumptions as the pervious tests and the runtime was 1 hour 16 minutes. The prediction time with the DNN model was 11 milliseconds for the entire set. Figure 5.5 plots the relative errors against the number of nodes.



(a) Total Network Capacity

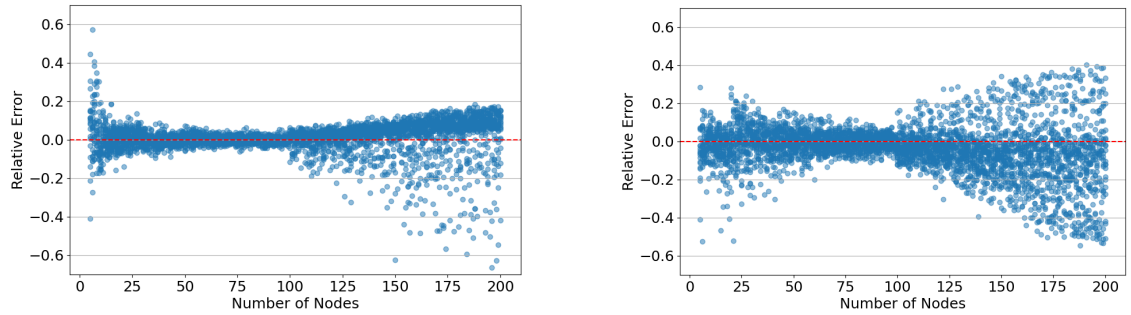
(b) Total Fiber Cost

Figure 5.5: Relative error against number of nodes for both outputs of the second DNN model (N in [5,100]).

From Figure 5.5 it was concluded that the average relative error is 2.47% for the total network capacity and 5.29% for the total fiber cost. It was also shown that for the total network capacity (Fig. 5.5(a)) 89.45% of the examples have a relative error below 5%, and 96.67% of the examples have a relative error below 10%. In the case of the total fiber cost (Fig. 5.5(b)) 87.02% of the examples have a relative error below 10%, and 94.24% of the examples have a relative error below 15%.

It can be seen that the model tends to have better performance for networks with a higher number of nodes, having a more irregular performance on networks with very few nodes. A possible explanation for this behavior is that smaller networks might exhibit more variability in their features and in the relationships between features and labels that make it more challenging for the model to predict accurately. On the other hand, larger networks could be more homogeneous, exhibiting more consistent patterns that the model can learn and predict more effectively.

An additional test set with 3920 networks was generated to evaluate the second model's performance on networks with a number of nodes outside the training range. This set was generated under the same conditions as the previous sets, but now with the number of nodes varying from 5 to 200. The runtime of *routing_nx_no_blocking* for this set was 55 hours 31 minutes. The predictions with the DNN model took only 79 milliseconds for the entire set. The scatter plot for this set of networks comparing the relative errors against the number of nodes is shown on Figure 5.6.



(a) Total Network Capacity

(b) Total Fiber Cost

Figure 5.6: Relative error against number of nodes for both outputs of the second DNN model (N in [5,200]).

The plots of Figure 5.6 show that the DNN tends to perform in a similar manner as in Fig. 5.5 when the nodes range between 5 and 100. For nodes outside this range, the performance becomes more inconsistent as the number of nodes grows. As in the first model, this is a case of OOD generalization [42], as the test data in the range of 101 to 200 nodes would be considered out-of-distribution, which justifies the model's unreliable performance. Nevertheless, based on the plots, we can see that the model still performs well up until networks with 115 to 120 nodes.

Table 5.10 shows the results of the application of the second DNN model and the routing with the

SDM solution to different random networks, comparing also the times taken to reach the results.

Table 5.10: Second model's predictions in random networks. y1: Total Network Capacity; y2: Total Fiber Cost

Number of Nodes	$y1$ (Tb/s)	Pred. $y1$ (Tb/s)	Relative Error $y1$ (%)	$y2$ (10^3 km)	Pred. $y2$ (10^3 km)	Relative Error y2 (%)	Routing time (s)	Pred. time (ms)
10	48.0	45.39	5.44	24.466	24.077	1.59	0.23	58.87
20	303.2	317.92	4.86	14.714	14.000	4.85	0.24	4.00
30	708.0	705.22	0.39	27.048	26.622	1.57	0.39	3.04
40	803.0	820.11	2.13	122.270	123.209	0.77	0.70	8.02
50	1244.2	1214.39	2.40	231.634	257.360	11.11	1.23	16.46
60	1837.2	1937.3	5.45	267.024	247.967	7.14	2.20	16.66
70	3432.8	3393.9	1.13	128.438	131.789	2.61	3.61	10.29
80	3185.0	3197.8	0.40	625.998	596.997	4.63	5.94	22.92
90	5898.6	5864.0	0.59	189.506	194.540	2.66	9.04	22.59
100	5394.6	5376.5	0.34	718.090	690.019	3.91	13.48	17.13

The results above show that the model tends to have a good performance in generated networks in this range of nodes, with the relative errors being generally low. The prediction times with the DNN model are always significantly faster than the computation with *routing_nx_no_blocking* (that difference being more obvious in networks with more nodes).

Table 5.11 shows the second DNN model's predictions and the SDM routing solution's results in the same real-world reference networks considered before.

Table 5.11: Second model's predictions in reference networks. y1: Total Network Capacity; y2: Total Fiber Cost

Network Name	$y1$ (Tb/s)	Pred. $y1$ (Tb/s)	Relative Error $y1$ (%)	$y2$ (10^3 km)	Pred. $y2$ (10^3 km)	Relative Error $y2$ (%)	Routing time (ms)	Pred. time (ms)
CESNET	32.6	37.22	14.16	5.343	5.167	3.29	186.91	68.87
COST239	81.2	82.83	2.01	24.06	23.07	4.11	170.72	17.93
DTAG	147.4	145.91	1.01	10.88	10.95	0.69	199.14	14.71
NSFNET	98.0	104.06	6.18	45.39	38.63	14.87	203.16	61.34
UBN	272.8	269.78	1.10	85.418	101.42	18.73	269.167	10.58

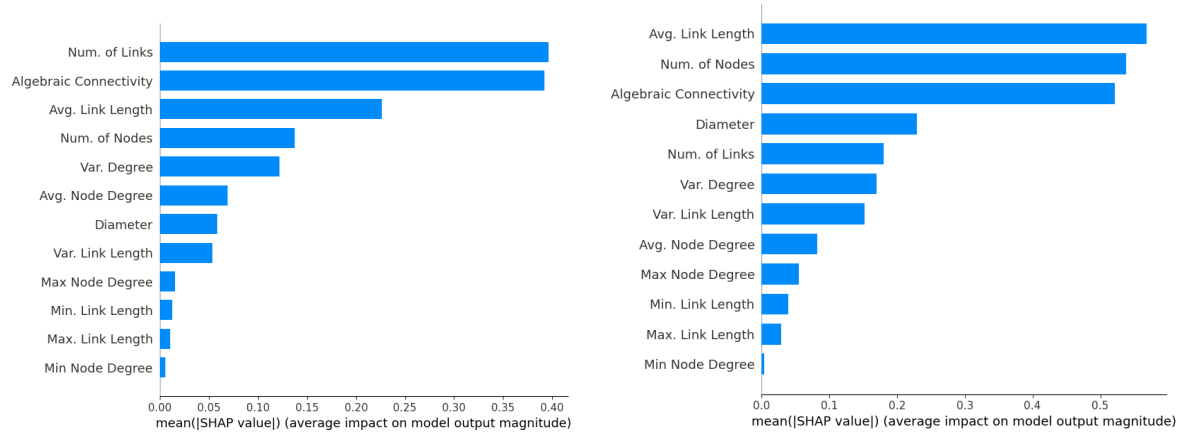
The evaluation on the reference networks reveals that the model delivers accurate predictions in the majority of cases. However, there are instances where higher relative errors have been observed, with three instances exceeding a 10% error rate. These discrepancies can likely be attributed to specific characteristics of these networks, which the model may not have effectively learned during its training phase. For example, the CESNET prediction is in the range where the model has a less consistent performance, being a network with 7 nodes. It's worth noting that, despite these occasional prediction inaccuracies, the model significantly reduces computation times when compared to the heuristic routing method, being always able to consistently predict in just a few milliseconds.

5.3 Importance of the Features to the Model's Predictions

In order to comprehend the behavior of the DNN models, it is important to understand the impact of each feature on the model's decision. To achieve this, an analysis of feature importance was conducted using the SHapley Additive exPlanations (SHAP) method [43]. This method assigns an importance value to each feature, with features that have a greater influence on a model's output being attributed higher SHAP values.

The SHAP method uses a game theory approach to determine feature importance, with each feature being considered a player, and the predictions being the payout. The SHAP value for a feature, representing its importance, is computed as its average contribution to the prediction across all possible combinations of features. These combinations are formed by keeping some features constant while randomizing others. This approach allows SHAP to accurately identify the features that most significantly influence the model's output, although it can be computationally expensive due to the need to evaluate all possible feature combinations [44].

The SHAP analysis was implemented using the Python library, *shap*, which provides an efficient and streamlined method for conducting the analysis and offers clear visualizations of the results. For the computation of the SHAP values, a random subset of 1000 instances was selected from the original set from which the training, validation and test data were obtained, for each respective model. The use of these smaller subsets allows for the computation times to be kept low, while still being ideal for this analysis, as they have the same data distribution that the model was trained on. Note that other data sets with the same data distribution of the training sets could also have been used (with the SHAP results being identical). The SHAP values for both outputs of the first DNN model are presented in Fig. 5.7.



(a) Total Network Capacity

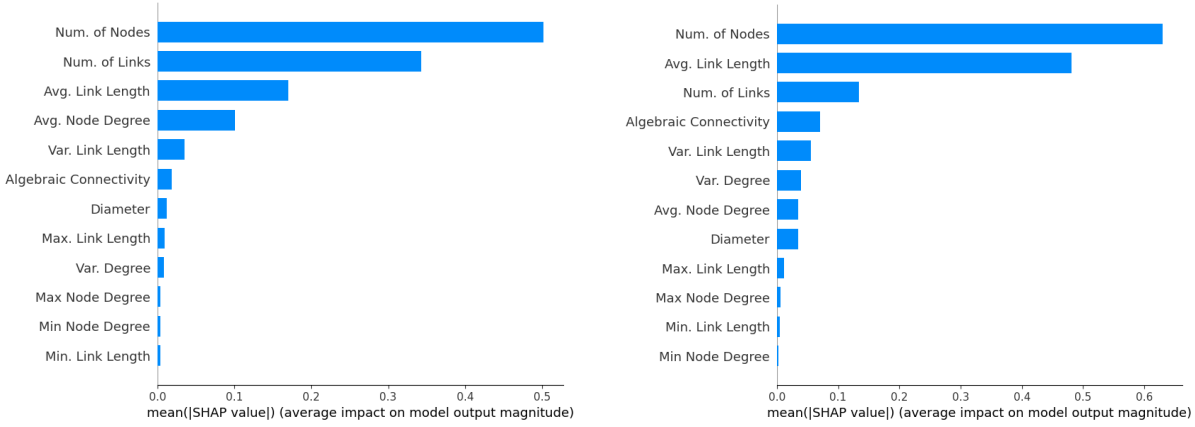
(b) Average Channel Capacity

Figure 5.7: SHAP analysis for the first DNN model.

From these results we can conclude that the number of links and the algebraic connectivity are the two most important parameters in the prediction of the total network capacity (Fig. 5.7(a)). Both of these parameters are indicative of the network's overall connectivity, as networks with a higher number of links and higher algebraic connectivity can be considered as being more connected and thus having more paths for the data to flow through. The fact that these parameters are more important than the number of nodes may be due to the fact that this output is being calculated in a capacitated routing setting, where blocking of traffic demands occurs. This indicates that the connectivity of the network, that is, the existence or not of many alternative paths, which in turn impacts directly the number of traffic demands being blocked, are the most important factors on the determination of the total network capacity in the first DNN model.

When it comes to the second output, the average channel capacity (Fig. 5.7(b)), the average link length is the most important parameter. Since the capacity of the optical channels was determined through the optical reach, this result is somewhat expected (as the length of the optical channels is necessarily related to the average link length of the network). Although we can also see that the number of nodes and the algebraic connectivity have a large impact on this output's prediction as well.

Figure 5.8 shows the SHAP results for the second DNN model.



(a) Total Network Capacity

(b) Total Fiber Cost

Figure 5.8: SHAP analysis for the second DNN model.

In this case we can see that the parameter with more influence on the prediction of the total network capacity (Fig. 5.8(a)) is now the number of nodes (followed by the number of links and the average link length). Since we are now in a setting of unconstrained routing (there is no blocking), it is normal that the overall size of the network (reflected on the number of nodes and links) has a large impact on this output (as a full-mesh logical topology is being considered). When it comes to the total fiber cost (Fig. 5.8(b)) the number of nodes, the average link length and the number of links are the features with

more importance. These results are expected given that the cost in km of fiber is heavily dependent on the network's size and the average length of it's links.

5.4 Chapter Conclusions

In this chapter, the training and testing of the two DNN regression models was detailed. It was shown that both models have generally strong performances, predicting total network capacity, average channel capacity, and total fiber cost with typically low relative errors. Furthermore, the time required for these predictions was significantly less than that needed to compute the routing solutions. The DNN models were able to make predictions in mere milliseconds, while routing could take several seconds for a single network. Even when analyzing large sets of networks, where routing algorithms could take hours to find a solution, the DNN models maintained their millisecond prediction times. This underscores the advantage of applying these ML techniques in the context of network capacity prediction.

The SHAP analysis revealed that the number of links and algebraic connectivity are the most important features when predicting the total network capacity in the first DNN model. These parameters reflect the network's overall connectivity, with more connected networks offering more alternative paths, which is particularly relevant in a capacitated routing setting. For the second output, average channel capacity, the average link length emerges as the most influential parameter, which is an expected result as the channel capacity was determined through the optical reach. In the second DNN model, where an unconstrained routing setting is considered, the overall network size, reflected by the number of nodes and links, greatly impacts the total network capacity. For the total fiber cost, the number of nodes, average link length, and number of links are the most important features. These results align with expectations, as the fiber cost in kilometers is heavily dependent on network size and average link length.

In certain instances, the DNN models showed a less consistent prediction capability. The first model's total network capacity predictions were generally less accurate compared to its average channel capacity output. The second model's predictions were less reliable for networks with fewer nodes. These issues could potentially be mitigated through a more comprehensive hyperparameter tuning process, including larger search spaces and the testing of alternative hyperparameters, such as different optimizer algorithms or loss functions.

Additionally, developing models with a more specific focus, that is, focused on predicting a single output or based on inputs that vary within a smaller range of values, could also bring better performances in these cases. By focusing on a single output (for example, a model with the only output being the total network capacity in a constrained routing scenario), the DNN model can optimize its weights and biases specifically for that output. This targeted optimization allows the model to learn the unique patterns and relationships associated with the output more effectively, potentially leading to more accurate and reliable predictions, and also simplifying the model's training process. On the other hand, by considering

inputs that vary within a smaller range of values (for example, a model focused on making predictions only for networks with a low number of nodes in the SDM scenario), the model can focus on the most relevant data patterns in that range, potentially improving its predictive performance.

Chapter 6

Conclusion

In this chapter, the conclusions of this thesis are presented, with the main accomplishments of this work being summarized. Suggestions for future developments are also stated.

6.1 Conclusions

In this thesis, two Deep Neural Network (DNN) models were successfully developed with the purpose of determining the capacity of optical backbone networks. The first model was designed to predict the total network capacity and average channel capacity considering a constrained number of optical channels per link due to the bandwidth limitations of fiber transmission. The second model was developed to estimate the unconstrained total network capacity and the total cost in kilometers of fiber deployed, under a scenario where an SDM solution is implemented to overcome said limitations. In both models a full-mesh logical topology was assumed, and the predictions were done based on input parameters related to the physical topology of the networks. These models were created with the objective of delivering accurate estimations with short computation times.

In Chapter 2, an overview of the fundamental concepts that underpin the work developed in this thesis is presented. It was shown that the physical topology of a telecommunications network can be represented by a graph and the flow of traffic in the network (the logical topology) can be represented by a traffic matrix. Various network parameters, relevant for the development of the DNN models, are defined and the concept of routing introduced, as well as a general strategy for solving routing problems. The definition of an optical channel is given and it is shown that, given the limitations in the WDM transmission bandwidth, there is a limit in the number of optical channels that can simultaneously be transmitted in an optical fiber. It was also seen that the channel capacity can be determined through the relation between optical reach and the Shannon channel capacity. This makes the process of determining the network capacity and average channel capacity more straightforward and computationally less

complex.

Also in Chapter 2, various state of the art applications of machine learning techniques in the context of optical networking are presented. These techniques cover applications related to the physical and network layers, including ones focused on the determination of the QoT and others focused on the routing and wavelength assignment process. In these various applications, neural network models have consistently shown to have the best generalization capabilities and reduced computation times.

In Chapter 3, the model for the generation of random networks implemented in this dissertation project is detailed. This model makes use of the same functioning principles as the Waxman model (including the spatial disposition of the nodes in a 2D plane and the probability of link establishment based on their relative distance), but changes were made to guarantee that the generated topologies more closely resemble real-world optical backbone networks. It was shown that the networks generated with the developed model follow the same essential characteristics of real-world networks: the networks have connected topologies which are resilient to single link failures (having an edge connectivity of at least 2), the node degree follows a Poisson distribution, and the link length distribution follows the Waxman link probability.

In Chapter 4, the two routing solutions developed in this work are presented: a constrained routing algorithm, where the limitation in the number of optical channels per fiber is assumed; and an unconstrained routing algorithm with fiber assignment, where that limitation is overcome through the addition of optical fibers to the network's links. These algorithms, which determine the RWA solution, are applied to the networks to determine the features and labels for the training and testing of the respective DNN model. In the case of unconstrained routing with the SDM solution, two fiber assignment algorithms were developed, one that focuses on providing fast computation times, and the other that focuses on making a more efficient fiber allocation. It was concluded that, despite the second algorithm's advantage in terms of efficient fiber placement, the faster first algorithm provides a better solution, given the need to apply it to a large number of networks. The application of these routing tools to random networks and reference networks allowed for the development of an analysis where it was shown that the network capacity growth is limited by the blocking of traffic demands, which increases as the networks grow larger in number of nodes (in full-mesh logical topologies). Through the SDM solution it was possible to achieve significantly higher network capacities, but at the cost of more fiber being used in the network.

Finally, in Chapter 5, the training and testing of the DNN models is presented. It was shown that these models tend to provide accurate results on both random networks as well as reference networks. When testing on a set of random networks, the first model's predictions for the total network capacity showed an average relative error of 6.17%, with 79.23% of the examples having a relative error below 10% and 90.81% of examples below 15%. The prediction of the average channel capacity showed an average relative error of 2.84%, with 83.36% of examples having a relative error below 5% and 97.6% below 10%.

In the second DNN model, the total network capacity predictions showed an average relative error of 2.47%, with 89.45% of the predictions having errors below 5% and 96.67% with errors below 10%. The total fiber cost was predicted with an average relative error of 5.29%, with 87.02% of the examples having errors below 10% and 94.24% below 15%. When testing on reference networks low relative errors were also achieved in most cases, further confirming the generated networks' similarities to real-world networks. Furthermore, it was seen that the DNN models were able to make the predictions in just a few milliseconds, even when considering sets with thousands of networks. This is a very significant advantage over the heuristic routing algorithm's computation times, which can take many hours in larger sets of networks.

Given the DNN models' overall accurate predictions and quick prediction times, and given that the model's inputs are physical topology characteristics of the networks, these DNN models could prove to be useful tools in the design of optical backbone networks, allowing for an immediate determination of the network capacity, average channel capacity or total fiber cost as the networks' physical topology is delineated.

6.2 Future Work

As mentioned on the Introduction, an alternative approach to achieve ultra-high bandwidth transmission was the consideration of BDM (specifically, multi-band transmission on the S+C+L bands). As a suggestion of possible future work, this solution could be explored. If the optical reach values for transmission in these optical bands are determined, the average channel capacity and total network capacity can be calculated with minimal modifications to the routing code developed in this thesis.

Investigating an alternative SDM solution is also a potential area for further research. In this work, the computation time was a restricting factor in the choice of the fiber assignment algorithm. If, instead of a complete elimination of the blocking of traffic, only a reduction was set as a goal, a more efficient fiber assignment algorithm could be employed, maintaining low computation times due to a less exhaustive fiber addition process. This approach would lead to a different type of network analysis, where both the blocking effect and fiber cost would need to be considered.

Another possibility for future developments based on this work is the exploration of other DNN models (or even other ML techniques). Through the generative graph model and the routing algorithms developed in this thesis it is easy to obtain a large amount of data to train and test ML models. This opens up the range of ML applications related to this type of optical networks, including, for example, techniques focused on improving OOD generalization performance, or models that make predictions based on the RWA solution (including parameters related to the link loads, path lengths, etc.), which may prove to be useful in the determination of the network capacity or other network parameters.

Bibliography

[1] P. J. Winzer and D. T. Neilson, “From scaling disparities to integrated parallelism: A decathlon for a decade,” *Journal of Lightwave Technology*, vol. 35, no. 5, pp. 1099–1115, 2017.

[2] J. Pires, “Chapter 1 - Introduction,” Optical Networks, Instituto Superior Técnico, 2023.

[3] N. Deng, L. Zong, H. Jiang, Y. Duan, and K. Zhang, “Challenges and Enabling Technologies for Multi-Band WDM Optical Networks,” *Journal of Ligthwave Technology*, vol. 40, no. 11, pp. 3385–3394, 2022.

[4] A. Freitas and J. Pires, “Investigating the Impact of Topology and Physical Impairments on the Capacity of an Optical Backbone Network,” *Photonics*, vol. 11, no. 4, 2024. [Online]. Available: https://www.mdpi.com/2304-6732/11/4/342

[5] F. Musumeci, C. Rottondi, A. Nag, I. Macaluso, D. Zibar, M. Ruffini, and M. Tornatore, “An Overview on Application of Machine Learning Techniques in Optical Networks,” *IEEE Communications Surveys and Tutorials*, vol. 21, no. 2, pp. 1383–1408, 2019.

[6] R. Gu, Z. Yang, and Y. Ji, “Machine learning for intelligent optical networks: A comprehensive survey,” *Journal of Network and Computer Applications*, vol. 157, no. 102576, 2020.

[7] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring Network Structure, Dynamics, and Function using NetworkX,” Proceedings of the 7th Python in Science Conference (SciPy 2008), 2008.

[8] pytorch.org, “PyTorch - About,” Available from online archive: https://web.archive.org/web/20180615190804/https://pytorch.org/about/, 2017, accessed on 20.04.2024.

[9] C. Pavan, R. M. Morais, J. R. F. da Rocha, and A. N. Pinto, “Generating Realistic Optical Transport Network Topologies,” *Journal of Optical Communication Networks*, vol. 2, no. 1, pp. 80–90, 2010.

[10] J. Pires, “Chapter 2 - Fundamentals of Networks,” High Speed Networks, Instituto Superior Técnico, 2023.

[11] K. Higashimori, T. Inoue, T. Tanaka, F. Inuzuka, and T. Ohara, “Impact of physical topology features on performance of optical backbone networks,” in *2022 International Conference on Optical Network Design and Modeling (ONDM)*, 2022, pp. 1–6.

[12] “NetworkX-algebaric_connectivity,” Available at: https://networkx.org/documentation/stable/reference/generated/networkx.linalg.algebraicconnectivity.algebraic_connectivity.html, 2023, accessed on 10.05.2024.

[13] FOwiki.com, “Understand Fiber Attenuation,” Available online: http://fowiki.com/b/understand-fiber-attenuation/, 2015, accessed on 27.05.2024.

[14] Ciena, “What is WDM or DWDM?” Available online: https://www.ciena.com/insights/what-is/What-Is-WDM.html, accessed on 27.05.2024.

[15] FiberLabs, “Wavelength-Division Multiplexing (WDM),” Available online: https://www.fiberlabs.com/glossary/wavelength-division-multiplexing/, accessed on 30.04.2024.

[16] R.-J. Essiambre and R. W. Tkach, “Capacity trends and limits of optical communication networks,” *Proceedings of the IEEE*, vol. 100, no. 5, pp. 1035–1055, 2012.

[17] A. Ferrari, A. Napoli, J. K. Fischer, N. Costa, J. Pedro, N. Sambo, E. Pincemin, B. Sommerkohrn-Krombholz, and V. Curri, “Upgrade Capacity Scenarios Enabled by Multi-band Optical Systems,” in *21st International Conference on Transparent Optical Networks (ICTON)*, Angers, France, 2019.

[18] A. Napoli, N. Calabretta, J. K. Fischer, N. Costa, S. Abrate, J. Pedro, V. Lopez, V. Curri, D. Zibar, E. Pincemin, S. Grot, G. Roelkens, C. Matrakidis, and W. Forysiak, “Perspectives of Multi-band Optical Communication Systems,” in *The 23rd OptoElectronics and Communications Conference (OECC 2018)*, 2018.

[19] A. Souza, B. Correia, N. Costa, J. Pedro, and J. Pires, “Accurate and scalable quality of transmission estimation for wideband optical systems,” in *2021 IEEE 26th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2021, pp. 1–6.

[20] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[21] R.-J. Essiambre, G. Kramer, P. J. Winzer, G. J. Foschini, and B. Goebel, “Capacity limits of optical fiber networks,” *Journal of Lightwave Technology*, vol. 28, no. 4, pp. 662–701, 2010.

[22] A. Gumaste and T. Antony. (2003) WDM Network Design. Accessed on 02.05.2024. [Online]. Available: https://www.ciscopress.com/articles/article.asp?p=30886&seqNum=4

[23] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[24] Y. Pointurier, "Machine learning techniques for quality of transmission estimation in optical networks," *Journal of Optical Communications and Networking*, vol. 13, no. 4, pp. B60–B71, 2021.

[25] A. F. Arruda, "Awareness of optical network performance with machine learning methods," Master's thesis, Instituto Superior Técnico, June 2022.

[26] R. M. Morais and J. Pedro, "Machine Learning Models for Estimating Quality of Transmission in DWDM Networks," *Journal of Optical Communications and Networking*, vol. 10, no. 10, pp. D84–D99, 2018.

[27] R. Proietti, X. Chen, A. Castro, G. Liu, H. Lu, K. Zhang, J. Guo, Z. Zhu, L. Velasco, and S. J. B. Yoo, "Experimental demonstration of cognitive provisioning and alien wavelength monitoring in multi-domain eon," in *Optical Fiber Communication Conference*. Optica Publishing Group, 2018, p. W4F.7. [Online]. Available: https://opg.optica.org/abstract.cfm?URI=OFC-2018-W4F.7

[28] I. Martín, S. Troia, J. A. Hernández, A. Rodríguez, F. Musumeci, G. Maier, R. Alvizu, and O. González de Dios, "Machine learning-based routing and wavelength assignment in software-defined optical networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 871–883, 2019.

[29] S. Chugh, S. Ghosh, A. Gulistan, and B. Rahman, "Machine Learning Regression Approach to the Nanophotonic Waveguide Analyses," *Journal of Lightwave Technology*, vol. 37, no. 24, pp. 6080–6089, 2019.

[30] S. Ghosh, "A comprehensive guide to data preprocessing," Available at: https://neptune.ai/blog/data-preprocessing-guide, 2023, accessed on 10.05.2024.

[31] Y. Lecun, "A theoretical framework for back-propagation," in *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, D. Touretzky, G. Hinton, and T. Sejnowski, Eds. Morgan Kaufmann, 1988, pp. 21–28.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[33] "Coefficient of determination, r-squared," Available: https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/regression-and-correlation/coefficient-of-determination-r-squared.html, accessed on 15-05-2024.

[34] J. Bashyal, “Absolute and relative error: Definition, formula, examples, differences,” Available: https://scienceinfo.com/absolute-and-relative-error-definition/, 2023, accessed on 15-05-2024.

[35] J. Bisht. (2022, 7) Erdos-Renyi Model for generating random graphs. Accessed on 08.05.2024. [Online]. Available: https://www.geeksforgeeks.org/erdos-renyl-model-generating-random-graphs/

[36] H. Zang, J. P. Jue, B. Mukherjee *et al.*, “A review of routing and wavelength assignment approaches for wavelength-routed optical WDM networks,” *Optical networks magazine*, vol. 1, no. 1, pp. 47–60, 2000.

[37] D. A. Moniz, “Design of spectral- and cost-efficient high-capacity optical transport networks,” PhD thesis, Instituto Superior Técnico, 2022.

[38] R. Matzner, D. Semrau, R. Luo, G. Zervas, and P. Bayvel, “Making intelligent topology design choices: understanding structural and physical property performance implications in optical networks,” *Journal of Optical Communications and Networking*, vol. 13, no. 8, pp. D53–D67, 2021. [Online]. Available: https://opg.optica.org/jocn/abstract.cfm?URI=jocn-13-8-D53

[39] D. Semrau, S. Durrani, G. S. Zervas, R. I. Killey, and P. Bayvel, “On the relationship between network topology and throughput in mesh optical networks,” *ArXiv*, vol. abs/2008.06708, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221140034

[40] V. Chugani, “Skewness be gone: Transformative tricks for data scientists,” 2024, accessed on 15-05-2024. [Online]. Available: https://machinelearningmastery.com/skewness-be-gone-transformative-tricks-for-data-scientists/

[41] J. Brownlee, “Hyperparameter optimization with random search and grid search,” 2020, accessed on 15-05-2024. [Online]. Available: https://machinelearningmastery.com/hyperparameter-optimization-with-random-search-and-grid-search/

[42] Z. Shen, J. Liu, Y. He, X. Zhang, R. Xu, H. Yu, and P. Cui, “Towards out-of-distribution generalization: A survey,” *CoRR*, vol. abs/2108.13624, 2021. [Online]. Available: https://arxiv.org/abs/2108.13624

[43] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[44] C. Molnar, “Interpretable machine learning - shapley values,” 2023, accessed on 17-05-2024. [Online]. Available: https://christophm.github.io/interpretable-ml-book/shapley.html

Appendix A

Additional Information About the Code

A.1 Executing the program routing_nx

Without command-line arguments

To run *routing_nx* without command-line arguments, the input file with the networks must be named *"graphs.gpickle"* and it must be in the working directory. When running the program this way, the following behavior will occur: the routing will be done in an uncapacitated way, the sorting order will be shortest-first, and the average channel capacity value will be determined considering the optical reach table for 64 Gbaud transmission (Table 2.1). The results will be output in the *"dnn_data.npy"* file and not printed in the terminal.

With command-line arguments

With command-line arguments the user can control how the program execution will be done. The possible arguments are:

-f, --file_name: The name of the file to load. Default is 'graphs.gpickle'.

-d, --directory: The directory of the file. Default is the current working directory.

-p, --print_mode: Can be either True or False. Set to True by adding '-p' to command-line inputs. In that case the results will be printed in the terminal and not saved in a file. Not putting -p defaults to False.

-m, --max_link_cap: Link capacity, an integer between 0 and 999999 that limits the number of optical channels per link. Default is 999999 (Uncap. Routing).

-s, --sorting_order: Sorting order. Can be "shortest", "longest" or "largest". Default is "shortest".

-b, –baud_rate: Baud Rate. Can be 64 or 128. Used to define the optical reach table to be used from Table 2.1. Default is 64.

So, for example, if the user wants to run a file named *my_graph.gpickle* that is in the working directory, do the capacitated routing with a link capacity of 75 and longest-first sorting, print the results in the terminal, and consider the optical-reach table for 128 Gbaud transmission, the code can be run with:

python routing_nx.py -f my_graph.gpickle -m 75 -s longest -p -b 128

A.2 Example of the results of *routing_nx* printed in the terminal

```
----PHYSICAL TOPOLOGY RESULTS----
Number of Nodes: 7
Number of links: 9
Minimum link length: 173.75
Maximum link length: 425.25
Average link length: 296.8388888888889
Variance of Link Length: 5980.669520987654
Minimum Node Degree: 2
Maximum Node Degree: 3
Average Node Degree: 2.5714285714285716
Variance of Node Degree: 0.24489795918367346
Network Diameter: 4
Algebraic Connectivity:268.84529340274526
```

(a) Physical Topology Results

```
----LOADS PER LINK / WAVELENGTHS PER LINK----
0-1 -> Load:2, Wavelengths:[1, 2]
0-2 -> Load:3, Wavelengths:[1, 2, 3]
0-6 -> Load:3, Wavelengths:[1, 2, 3]
1-0 -> Load:2, Wavelengths:[1, 2]
1-2 -> Load:1, Wavelengths:[1]
2-0 -> Load:3, Wavelengths:[1, 2, 3]
2-1 -> Load:1, Wavelengths:[1]
2-3 -> Load:3, Wavelengths:[1, 3, 2]
3-2 -> Load:3, Wavelengths:[1, 3, 2]
3-4 -> Load:2, Wavelengths:[1, 2]
3-5 -> Load:3, Wavelengths:[1, 2, 3]
4-3 -> Load:2, Wavelengths:[1, 2]
4-5 -> Load:3, Wavelengths:[1, 3, 2]
5-3 -> Load:3, Wavelengths:[1, 2, 3]
5-4 -> Load:3, Wavelengths:[1, 3, 2]
5-6 -> Load:3, Wavelengths:[1, 2, 3]
6-0 -> Load:3, Wavelengths:[1, 2, 3]
6-5 -> Load:3, Wavelengths:[1, 2, 3]
```

(b) Load and Wavelengths in the Links

```
----ROUTING RESULTS----
Minimum path length: 173.75
Maximum path length: 1350.9499999999998
Average path length: 451.87399999999997
Average Number of Hops per Demand: 1.7619047619047619
Average Number of Hops per Path (after routing): 1.533333
Minimum Link Load: 1
Maximum Link Load: 3
Average Load per Link: 2.5555555555555554

----BLOCKING----
Blocked Traffic: 12
Blocking Probability: 0.2857142857142857
----BLOCKED PATHS----
'source':1,'destination':3,'path':[[1, 2, 3]]
'source':3,'destination':1,'path':[[3, 2, 1]]
'source':0,'destination':3,'path':[[0, 1, 2, 3]]
'source':3,'destination':0,'path':[[3, 2, 1, 0]]
'source':2,'destination':4,'path':[[2, 3, 4]]
'source':4,'destination':2,'path':[[4, 3, 2]]
'source':0,'destination':4,'path':[[0, 1, 2, 3, 4]]
'source':4,'destination':0,'path':[[4, 3, 2, 1, 0]]
'source':5,'destination':1,'path':[[5, 6, 0, 1]]
'source':1,'destination':5,'path':[[1, 0, 6, 5]]
'source':4,'destination':1,'path':[[4, 3, 2, 1]]
'source':1,'destination':4,'path':[[1, 2, 3, 4]]


----CAPACITY RESULTS----
Average Channel Capacity (Gb/s): 576.1904761904761
Total Network Capacity (Gb/s): 17285.714285714283
```

(c) Routing Results

```
---- TRAFFIC DEMANDS PATHS AND WAVELENGTHS ---
0-1 -> Path:[0, 1], Wavelength:1
0-2 -> Path:[0, 2], Wavelength:1
0-5 -> Path:[0, 2, 3, 4, 5], Wavelength:2
0-6 -> Path:[0, 6], Wavelength:1
1-0 -> Path:[1, 0], Wavelength:1
1-2 -> Path:[1, 2], Wavelength:1
1-6 -> Path:[1, 0, 6], Wavelength:2
2-0 -> Path:[2, 0], Wavelength:1
2-1 -> Path:[2, 1], Wavelength:1
2-3 -> Path:[2, 3], Wavelength:1
2-5 -> Path:[2, 3, 5], Wavelength:3
2-6 -> Path:[2, 0, 6], Wavelength:3
3-2 -> Path:[3, 2], Wavelength:1
3-4 -> Path:[3, 4], Wavelength:1
3-5 -> Path:[3, 5], Wavelength:1
3-6 -> Path:[3, 5, 6], Wavelength:2
4-3 -> Path:[4, 3], Wavelength:1
4-5 -> Path:[4, 5], Wavelength:1
4-6 -> Path:[4, 5, 6], Wavelength:3
5-0 -> Path:[5, 4, 3, 2, 0], Wavelength:2
5-2 -> Path:[5, 3, 2], Wavelength:3
5-3 -> Path:[5, 3], Wavelength:1
5-4 -> Path:[5, 4], Wavelength:1
```

(d) Traffic Demands' Paths and Wavelengths (truncated results)

Figure A.1: Example of *routing_nx* results printed in the terminal.

A.3 Additional Algorithms of routing_nx

A.3.1 Shortest-Paths Function

The function *shortest_paths* is the function that determines all the shortest-paths between every pair of nodes. It receives as input a *NetworkX* Graph and outputs a list of lists, where each inner list contains dictionaries. Each dictionary represents a traffic demand between two nodes in the graph *G*. Note that *G* is a weighted graph where the weights are the link lengths. The dictionary contains the following keys:

- 'source': the source node;
- 'destination': the target node;
- 'distance': the total length of the path;
- 'path': a list of all shortest-paths between the source and target node.

To index this structure its possible to do *shortest_paths_list[i][j]*, where *i* is the index of the source node and *j* is the index of the target node in the nodes of *G*. This will give a dictionary representing the shortest-paths from the *i*-th node to the *j*-th node in the graph, along with their total length. If *i* and *j* are the same, the distance will be 0 and the path will be a list containing a single list with the node itself.

The pseudo-code for the *shortest_paths* function is presented below:

Algorithm 4: Shortest-Paths

Input: A *NetworkX* Graph, *G*.
Output: A list of the shortest-paths between all pairs of nodes, *all_paths*.

```
1:  Initialize an empty list all_paths
2:  for each node in G do                                   ▷ Considered as source node
3:      Initialize an empty list source_paths
4:      for each node in G do                               ▷ Considered as target node
5:          if source is not equal to target then
6:              paths ← nx.all_shortest_paths(G, source, target, weight = link length) ▷ Compute all
                shortest paths from source to target and store them in a list
7:              Compute the length of the paths (sum of the link lengths) and store it in total_length
8:              Append to source_paths a dictionary with 'source', 'destination', 'distance', and 'path'
9:          else
10:             Append to source_paths a dictionary with 'source', 'destination', 'distance' as 0, and 'path'
                as a list with only the source node
11:     Append source_paths to all_paths
12: return all_paths
```

A.3.2 Order Paths Function

The function *orderPaths* sorts the traffic demands according to the sorting order. It takes as input the output of *shortest_paths* (list of lists, where each inner list contains dictionaries) and the sorting order ('shortest', 'longest' or 'largest'). It outputs a list of dictionaries, where each dictionary represents a traffic demand between two nodes in the graph. The dictionaries contain the following keys:

- 'source': the source node;
- 'destination': the target node;
- 'distance': the total length of the path;
- 'path': a list of all shortest-paths between the source and target node;
- 'traffic': the traffic between the source and target nodes, as given by the *traffic_matrix*;
- 'hops': the number of hops between the source and target nodes, as given by the *hop_matrix*;
- 'routed': a boolean value indicating whether the traffic demand has been routed (set to False in this function).

The traffic demands are ordered according to the order parameter. If order is "shortest", the paths are sorted by distance in ascending order. If order is "longest", the paths are sorted by distance in descending order. If order is "largest", the paths are sorted by traffic in descending order. Paths with the same source and destination or with no traffic are excluded from the output.

Algorithm 5: Order Paths

Input: the list of lists with the shortest-paths (*paths*); the traffic matrix (*traffic_matrix*); the hop matrix (*hop_matrix*) and a string with the sorting order (*order*).
Output: a list with the ordered traffic demands, *ordered_paths*.

```
Initialize an empty list: path_list
for each path in paths do
    for each p in path do                ▷ Iterate over all shortest-paths in the list of lists structure
        Assign traffic and hops values from the matrices to p
        p["routed"] ← False
        if p["source"] is the same as p["destination"] OR p["traffic"] is 0 then
            Skip this path
        Get the possible paths from p
        if there are multiple possible paths then
            Find the path with lowest number of hops and remove paths that have more hops than
            that path        ▷ Between multiple shortest-paths, exclude the ones with more hops
        Append p to path_list
if order is "shortest" then
    Sort path_list by distance in ascending order
else if order is "longest" then
    Sort path_list by distance in descending order
else if order is "largest" then
    Sort path_list by traffic in descending order
return ordered_paths
```

A.3.3 Breaking a Path Tie

The *breakTie* function is called when there are two or more possible shortest paths between a pair of nodes. This situation happens when the *shortest_paths* function returns various same length shortest-paths between a given source and destination. This function receives as input *p* the current traffic demand and the *load_matrix* and it will return the path's index in *p['path']* for which routing the traffic to minimizes the load in the most loaded link. So, this function's goal is to find, between all possible paths, the one with the minimum most loaded link, which corresponds to the path where the load in the most loaded link is minimized. If there is a tie between all the most loaded links (a minimum can't be found), the function goes on to the second most loaded link, and so on. If by the end, a minimum most loaded link can't be found, that means that, between all paths considered, the link loads are the same, and so it is indifferent which path is chosen, and the first one in the list is picked. This function is implemented as follows:

Algorithm 6: BreakTie

Input: The currently considered traffic demand, *p*, and the *load_matrix*.
Output: The index in *p['path']* of the chosen path, *chosen_path*.

```
for each path in p['path'] do          ▷ Get a list of lists where each inner list has all loads of a path
    for each edge in the path do
        Append the load of the edge to loads_of_path.
    Copy loads_of_path to the corresponding index in all_loads.
    Clear loads_of_path.
Initialize an empty list maximum_list.
while True do                          ▷ Find the path with the minimum maximum load
    for each path_load in all_loads do ▷ Find the maximum load in each path
        if path_load is empty then
            Set chosen_path to 0.
            return chosen_path. ▷ If, by the end, the list is empty, that means that all maximum loads
                                  are the same, just select path 0
        Append the maximum load in path_load to maximum_list.
        Remove the maximum load from path_load.
    if all elements in maximum_list are equal then
        Clear maximum_list.
        Continue to the next iteration of the while loop.
    else                               ▷ If all elements are not the same then there's a minimum
        Set min_val to the minimum value in maximum_list.
        if min_val occurs only once in maximum_list then
            Set chosen_path to the index of min_val in maximum_list.
            return chosen_path.        ▷ Found the path with link with minimum maximum load
        else                                                          ▷ Minimum not unique
            Clear maximum_list.
            Continue to the next iteration of the while loop.
```

A.3.4 Wavelength Assignment

The *assign_wavelengths* function implements the first-fit wavelength assignment strategy. By the end of this process, each wavelength is represented as a different integer starting in 1 and up to a given limit, corresponding to the maximum number of optical channels per fiber. As the wavelength assignment process is done during the routing (in the *route* function), the order in which the wavelength assignment process is done will always correspond to the routing order. The limitation on the maximum number of wavelengths will also affect the routing process, as it can lead to the blocking of traffic demands due to wavelength unavailability.

The *assign_wavelengths* function has as inputs: the current traffic demand (*p*), the index in *p['path]* of the path chosen to route through (*chosen_path*), and the matrices to be updated (*path_wavelength_matrix* and *link_wavelength_matrix*). It outputs a boolean value that indicates if the wavelength assignment was successful (*True*) or not (*False*). If *False* is returned, that means that a wavelength unavailability situation has occurred, and so that traffic demand is then blocked in the *route* function. The *assign_wavelengths* function is implemented as follows:

Algorithm 7: Assign Wavelengths

Input: The current traffic demand (*p*), the index in *p['path]* of the path chosen (*chosen_path*), and the matrices *path_wavelength_matrix* and *link_wavelength_matrix*.
Output: Boolean value that indicates if the wavelength assignment was successful (True) or not (False).

```
Define all_link_wavelengths as empty set
for each link (between nodes x, y) of the path do ▷ Iterate over all links of the path and save all the wavelengths already present
    Update all_link_wavelengths with value in link_wavelength_matrix[x][y]
w ← 1
while w in all_link_wavelengths do
    w ← w + 1 ▷ Increment until it's found a number not in the set, to get the lowest possible wavelength not used in any link of the path
if w > MAX_WAVELENGTHS then
    return False
for each link (between nodes x, y) of the path do ▷ Iterate over all links of the path and add the new wavelength
    append w to link_wavelength_matrix[x][y]
path_wavelength_matrix[p['source']][p['destination']] ← w
return True
```

A.3.5 Routing a single Traffic Demand

The *route_path* function implements the routing of a traffic demand, given a chosen path. It does so by updating the *load_matrix* and *path_matrix* accordingly (updates all the links in the *load_matrix* according to the path and saves the path in the respective position of the *path_matrix*). This function also checks, during the process of updating the *load_matrix*, if the load in a given link has reached the defined link

capacity (*MAX_LINK_CAP*). If that is the case, the flag variable *update_net_flag* is set to *True* to indicate that the network needs updating and the saturated links are added to a list of links to be removed. Later on in the *route* function, the *update_network* function will remove the links and determine new shortest paths.

In the case where a traffic demand has traffic greater than one unit of traffic, it is necessary to see if the traffic can be accommodated by all the links of the chosen path (defined by the link's residual capacity) before updating the *load_matrix* and *path_matrix*. If it cannot be accommodated, that traffic demand will be blocked, and so the assigned wavelengths are removed from the respective links in *link_wavelength_matrix* and from *path_wavelength_matrix* and the variable *blocked_flag* is set to *True*, to indicate that traffic demand needs to be blocked in the *route* function. Note that this process is not necessary when one unit of traffic is considered, as in the way the routing is made, at this point of the routing, the residual capacity in all links of a chosen path is at least one.

The *route_path* function takes as input the current traffic demand to route (*p*), the *load_matrix*, the *path_matrix*, the *path_wavelength_matrix*, the *link_wavelength_matrix*, the index in *p['path]* of the path chosen (*chosen_path*) and an auxiliary variable (*p_aux*) that always refers to the original traffic demand (even after *p* is set to the one returned by *update_network*) to keep track of the original traffic value. It outputs a flag to set the network to be updated (*update_network_flag*), the links to be removed (*links_to_remove*), and a flag to set the traffic demand to be blocked (*blocked_flag*). The pseudo-code for this function is presented below:

Algorithm 8: Route Path

Input: The traffic demand (*p*), the *load_matrix*, *path_matrix*, *path_wavelength_matrix*, *link_wavelength_matrix*, the index in *p['path]* of the chosen path (*chosen_path*), and *p_aux*.
Output: *update_network_flag*, *links_to_remove* and *blocked_flag*

update_net_flag, blocked_flag set to *False* and *links_to_remove* initialized as empty list
if *p_aux['traffic']* > 1 **then**
    for each link (between nodes x, y) of the path **do**
        if *load_matrix[x][y] + p_aux['traffic']* $>$ *MAX_LINK_CAP* **then** ▷ *not enough capacity*
            blocked_flag = True
    if *blocked_flag* is *True* **then**
        Remove wavelength from *path_wavelength_matrix* and *link_wavelength_matrix*
        path_matrix[p['source']][p['destination']] $\leftarrow 0$
        return *update_net_flag, links_to_remove, blocked_flag*
path_matrix[p['source']][p['destination']] $\leftarrow$ *p['path'][chosen_path]*
for each link (between nodes x, y) of the path **do**
    load_matrix[x][y] $\leftarrow$ *load_matrix[x][y] + p_aux['traffic']*
    if *load_matrix[x][y] = MAX_LINK_CAP* **then**
        update_net_flag $\leftarrow$ True
        append link (x, y) to *links_to_remove*
return *update_net_flag, links_to_remove, blocked_flag*

A.3.6 Updating the Network

The *update_network* function is called whenever a link has become congested and needs to be removed (signaled in the *route* function by the *update_net_flag*). It receives as inputs a list of links to remove (*links_to_remove*) and the graph, G_{copy}. This function will recalculate the shortest-paths (by calling function *shortest_paths*, in Algorithm 4) and output the list of new shortest-paths. From that point on, in the *route* function, the new shortest-paths will be the ones considered in the routing process (signaled by *new_paths_flag*). This function is implemented as follows:

Algorithm 9: Update Network

Input: *links_to_remove* and the graph of the network, G_{copy}.
Output: recalculated shortest-paths in list of lists with dictionaries structure (same as in section A.3.1), *new_shortest_paths*.

```
for each link in links_to_remove do
    G_copy[link[0]][link[1]]['length'] ← 99999        ▷ Virtually remove the link from G_copy
new_shortest_paths ← shortest_paths (see Algorithm 4)
for each path in new_shortest_paths do
    for each p in path do        ▷ Iterate over list of lists (same as described in section A.3.1)
        if there are multiple possible paths then
            Find the shortest path in number of hops and remove paths that are longer than the
            shortest path
return new_shortest_paths
```

A.4 Example of Capacitated Routing Application

Figure A.2 depicts a test network used to exemplify the use of the constrained routing algorithm, as implemented in the *routing_nx* program. A logical full-mesh topology was considered and the sorting strategy used was the shortest-first. The link capacity considered was 5 units of traffic. The process of wavelength assignment as well as the possibility of blocking due to wavelength unavailability was ignored in this example, for the sake of simplicity. So this example focuses only on the assignment of the paths (π_k) associated to the traffic demands.

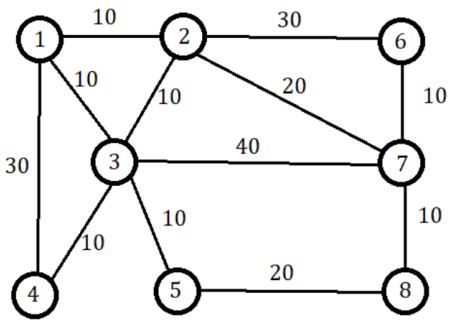


Figure A.2: Test Network - Physical Topology with the link lengths.

The first 46 traffic demands (the ones with distances 10, 20, 30 and four with distance 40) are routed through the following paths: [1, 2], [2, 1], [1, 3], [3, 1], [2, 3], [3, 2], [3, 4], [4, 3], [3, 5], [5, 3], [6, 7], [7, 6], [7, 8], [8, 7], [1, 3, 4], [4, 3, 1], [1, 3, 5], [5, 3, 1], [2, 3, 4], [4, 3, 2], [2, 3, 5], [5, 3, 2], [2, 7], [7, 2], [4, 3, 5], [5, 3, 4], [5, 8], [8, 5], [6, 7, 8], [8, 7, 6], [1, 2, 7], [7, 2, 1], [2, 6], [6, 2], [2, 7, 8], [8, 7, 2], [3, 2, 7], [7, 2, 3], [3, 5, 8], [8, 5, 3], [5, 8, 7], [7, 8, 5], [1, 2, 6], [6, 2, 1], [3, 2, 6] and [6, 2, 3].

The residual capacity of the network after the routing of these demands is represented in figure A.3.

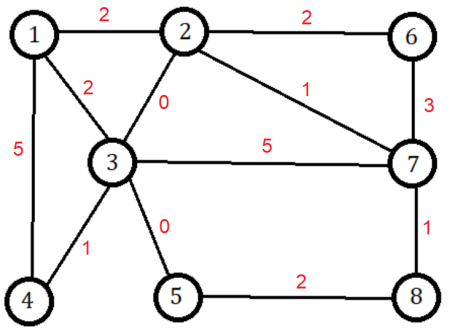


Figure A.3: Test Network - Residual Capacity represented bidirectionally (1).

The next traffic demand to route is t(4,7) with the path [4,3,2,7], but since the link (3,2) has a residual capacity of zero, an alternative routing path is determined: [4,3,7]. The computation of this path now

excludes the saturated link (3,2). The same process is done for the traffic demand t(7,4).

Afterwards the traffic demand t(4,8) is routed. Normally this would be done through the shortest path [4,3,5,8], but since the link (4,3) has now (after routing t(4,7)) residual capacity zero, an alternative path is computed: [4,1,2,7,8]. The same occurs for the symmetric demand t(8,4).

After the routing of these paths, the residual capacity of the network is now represented in figure A.4.

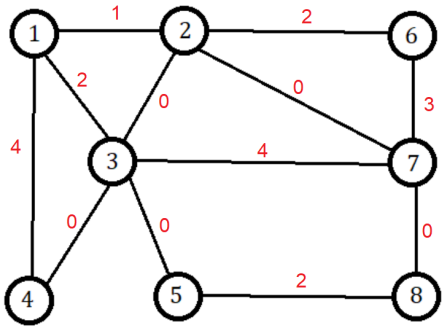


Figure A.4: Test Network - Residual Capacity represented bidirectionally (2).

The next traffic demand to route is t(5,6). This would normally be routed through the path [5, 8, 7, 6], but since the residual capacity of link (8,7) is zero, an alternative path needs to be determined. However, since any alternative path to this one would necessarily include link (5,3) and this link's residual capacity is also zero, it is impossible to route the traffic demand t(5,6). This traffic is then blocked. The same thing happens to the symmetric demand t(6,5).

The next traffic demands to route are t(1,8) and t(8,1). Given the residual capacities of the network, it is also impossible to establish a path between these nodes, and so this traffic is also blocked.

Finally, the last traffic demands t(4,6) and t(6,4) are routed through paths [4, 1, 2, 6] and [6, 2, 1, 4]. The loads in each of the links at the end of the routing process are represented in figure A.5. The total amount of blocked traffic is 4.

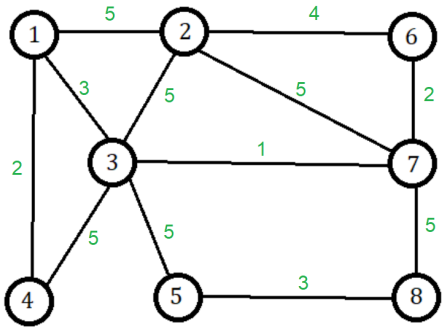


Figure A.5: Test Network - Loads in each link.

A.5 Alternative Strategy for Wavelength and Fiber Assignment

With the goal of better optimizing the addition of fibers over the one described in Algorithm 3, an alternative fiber assignment strategy was implemented and tested. In this alternative strategy, fiber and wavelength assignment are made simultaneously during the routing process (and not afterwards, as in the previous strategy) and so the function *assign_wavelengths_and_fibers* replaces *assign_wavelengths* in the *route* function. This results in a different approach, where instead of counting the number of fibers that need to be attributed given the traffic demands present in each link, the fibers are added during the routing and the wavelengths are assigned so as to minimize the need for adding more fibers. The pseudo-code for this strategy is presented below:

Algorithm 10: Wavelength and Fiber Assignment (Alternative Strategy)

Input: p, chosen_path, path_wavelength_matrix, link_wavelength_matrix and max_wl_per_fiber.
Output: Updated path_wavelength_matrix and link_wavelength_matrix.

```
Initialize tested_combinations as empty list
for r in range {1, ..., len(p['path'][chosen_path])} do ▷ Consider all possible combinations of adding fibers to links, starting with 0 fibers added until a fiber is added to all links of a path.
    link_combinations ← all combinations of r links
    for links in link_combinations do
        Add (in the link_wavelength_matrix) a fiber to the link or links in links
        fiber_combinations ← all combinations of fibers from all existing fibers of each link using the values in link_wavelength_matrix (fibers that have no space for additional wavelengths are excluded from the combinations)
        Filter out tested_combinations from fiber_combinations ▷ Avoid repeated computations.
        for fibers in fiber_combinations do ▷ Consider each combination of fibers from different links to check if there is a wavelength that fits.
            w ← CHECK_WAVELENGTHS(fibers)
            if w is not None then ▷ If a w that fits a given set of fibers is found
                path_wavelength_matrix[p['source']][p['destination']] ← w
                return True
            Append fibers to tested_combinations
        Remove the fibers added to the links to test next combination of fiber addition

▷ Function to find the wavelength that fits in a given combination of fibers. ◁
function CHECK_WAVELENGTHS(fibers,w_range = {1, ..., max_wl_per_fiber})
    all_wavelengths ← set of all the wavelengths in all links of a path
    possible_wavelengths ← set of all possible wavelengths in w_range
    available_wavelengths ← possible_wavelengths − all_wavelengths
    if available_wavelengths is empty then
        return None
    w ← min(available_wavelengths)
    for fiber in fibers do ▷ Add wavelength to all fibers in the path.
        Append w to fiber
    return w
```

In this strategy, for each traffic demand being routed, all combinations of addition of a fiber to each link of that traffic demand's path are tested. So this process is done on a per traffic demand basis (as

the function *assign_wavelengths_and_fibers* is called inside the main loop of *route*). It starts with the case of adding no new fiber to any link, then moving on to the case with the combinations of adding a single fiber to a link, then to the case with the combinations of adding two fibers to two links, etc., until the limit case where a fiber is added to every single link of a path. For each case of fiber addition, the combinations of fibers from the different links (including the already existing fibers and the added fibers) are generated. For each one of these combinations the *check_wavelenghs* function is called to check if it is possible to find a wavelength (w) that fits the fibers being considered (assuring wavelength continuity). If a wavelength is found then the process stops and the wavelength is added in the respective positions of the *path_wavelength_matrix* and *link_wavelength_matrix*. If not, then the next combination of fibers is tested, and then the next combination of fiber addition. In the limit case of no wavelength being found after various fibers being added, in the last case, a fiber is added to each link of the path, and so a wavelength is then guaranteed to be found, being thus impossible to have blocked traffic demands.

In this strategy, the fibers in each link as well as the wavelengths in each fiber are represented in the *link_wavelength_matrix*, which has a different structure than in Algorithm 3. The *link_wavelength_matrix* is now a matrix where each position has a list of lists, each inner list representing a different fiber, and each fiber has multiple integers (from 1 to *max_wl_per_fiber*) representing the different wavelengths. So, the list in position (i, j) of the *link_wavelength_matrix* represents the various wavelengths across different fibers in the link between nodes i and j. After the routing process, a *fiber_link_matrix* equivalent to the one used in Algorithm 3 is determined from the *link_wavelength_matrix*, being then used to determine the metrics related to the fibers (total number of fibers in the network, total fiber cost in km, etc.).

This method assures that the number of additional fibers is minimized on each path, as before the addition of any new fiber, each combination of existing fibers is considered. And when new fibers are added, all possibilities of addition between all links are considered, before moving on to adding more fibers. While this does not guarantee that the number of fibers is minimized network wide (as that would require an even more complex algorithm that requires the repetition of the routing with each fiber combination), tests show that it improves on the results of Algorithm 3. However, the computation times became very long, especially in larger networks.

A.5.1 Application Example

To better understand the functioning of Algorithm 10 and see its advantage in terms of efficiency, a simple example is presented. Figure A.6 represents part of a network, with the only nodes represented being A, B, C and D, as well as the respective links, with a single fiber each. In this example each line between the nodes represents a fiber, but note that each communication direction should have its own fiber. For simplicity, the process described here considers the transmission in a single direction, but the process is the exact same in the other communication direction.

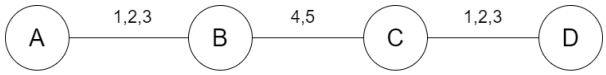


Figure A.6: Algorithm 10 Example - Starting Point.

Assuming that, after a certain point in the routing process, the wavelengths present in each fiber are the ones represented (as integers) above the links (resultant from the routing of traffic demands between other nodes in the network that are not represented and whose paths make use of these links). Considering also that the maximum number of optical channels per fiber (*max_wl_per_fiber*) is 5; if the next traffic demand to be routed is the one between nodes A and D ($t_{A,D}$) and the shortest path associated to this traffic demand is [A,B,C,D], the process of wavelength and fiber assignment described in Algorithm 10 will proceed as follows:

(1) Try to find a wavelength (integer between 1 and *max_wl_per_fiber*) that fits on the already existing fibers (respecting wavelength continuity and the principle that two optical channels with the same wavelength cannot co-exist on the same fiber). In this case the fiber combination tried is [$l_{A,B}f1$; $l_{B,C}f1$; $l_{C,D}f1$] (with $l_{X,Y}fn$ representing the fiber being considered, that is, in link between nodes X and Y, fiber n). It is found that there is no wavelength that fits these fibers (as all wavelengths between 1 and 5 are used across the links of the path between A and D).

(2) All combinations of addition of a single fiber to the path's links are tested. That process starts with the addition of a fiber to the link(A,B): $l_{A,B}f2$. The combination [$l_{A,B}f2$; $l_{B,C}f1$; $l_{C,D}f1$] is tested. Again, despite the newly added fiber, $l_{A,B}f2$, it is still not possible to find a wavelength that fits (as wavelengths 1 through 5 are used in $l_{B,C}f1$; $l_{C,D}f1$).

(3) The next fiber combination is tested. That is, a fiber is added to link(B,C), $l_{B,C}f2$ (note that $l_{A,B}f2$ is now removed, as the combinations being tested are of adding a single fiber). The test is made with the fiber combination [$l_{A,B}f1$; $l_{B,C}f2$; $l_{C,D}f1$]. With this new fiber it is now possible to find a wavelength that fits: wavelength 4. Since a solution was found, the process is complete and the next traffic demand will be routed (the main loop of Algorithm 2 continues on). The results of this process are presented in Fig. A.7.

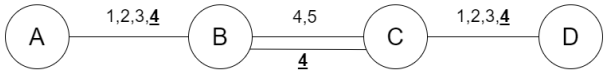


Figure A.7: Algorithm 10 Example - End Result.

This type of behavior can only happen if the fiber assignment is done together with the wavelength assignment. If this example were to be done with Alg. 3, two additional fibers would have been added to two links (A-B and C-D), instead of just one. However, it is important to note that, as the routing process advances, the paths being considered get longer (more links), the links tend to have more fibers, and each fiber more wavelengths. This makes the procedure of examining all combinations very slow.

Appendix B

Additional Figures

B.1 Topology of Reference Networks

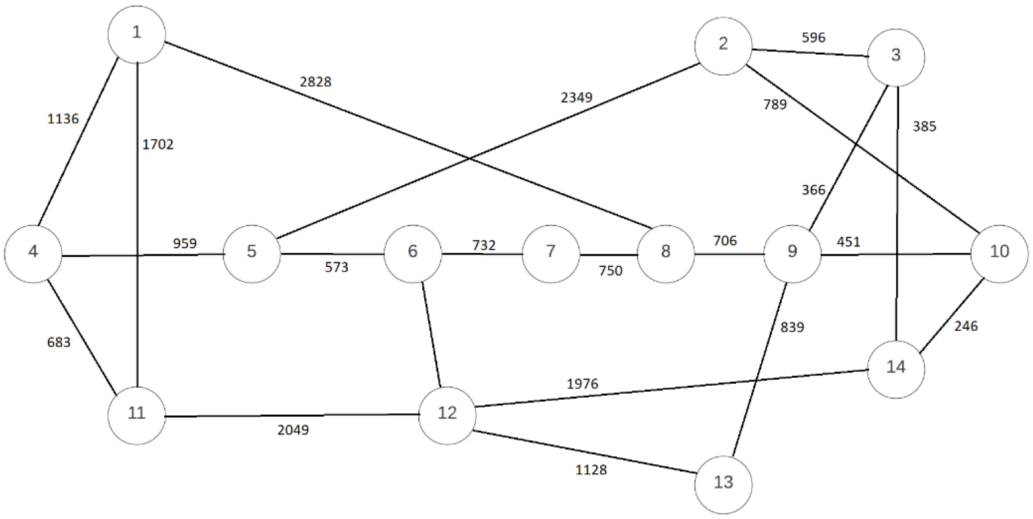


Figure B.1: NSFNET.

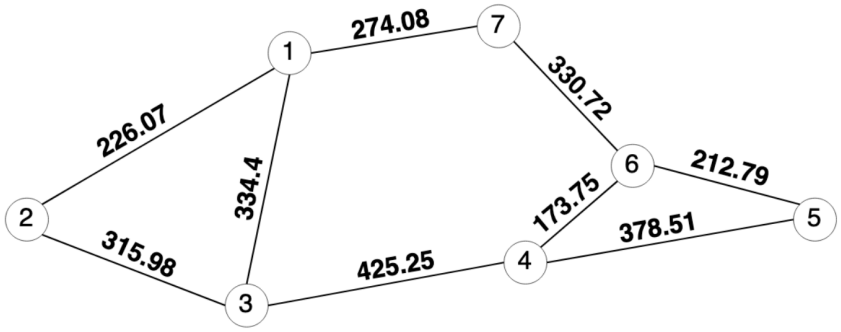


Figure B.2: CESNET.

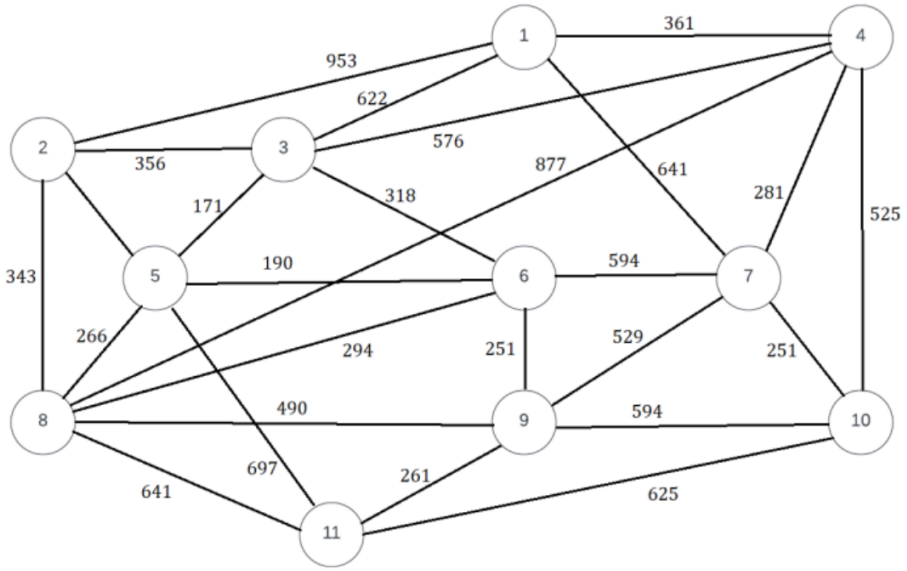


Figure B.3: COST239.

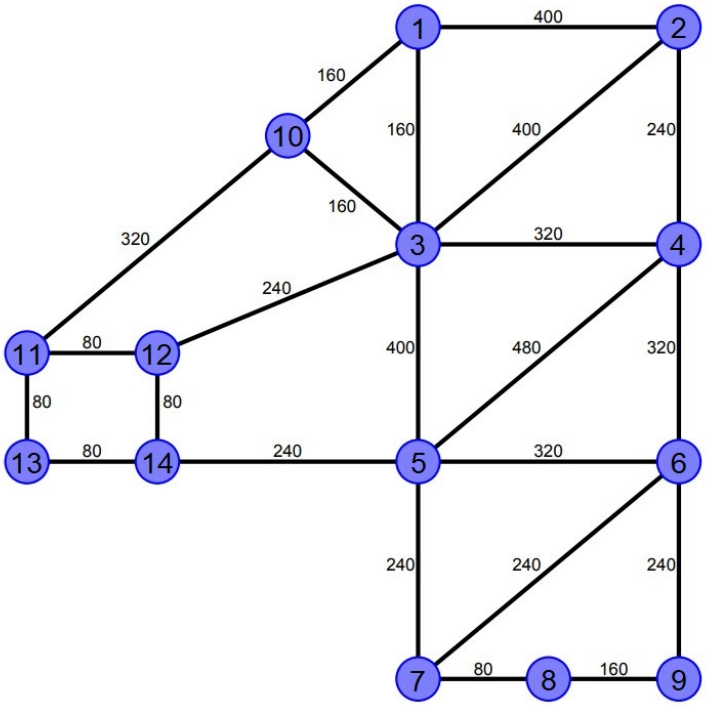


Figure B.4: DTAG.

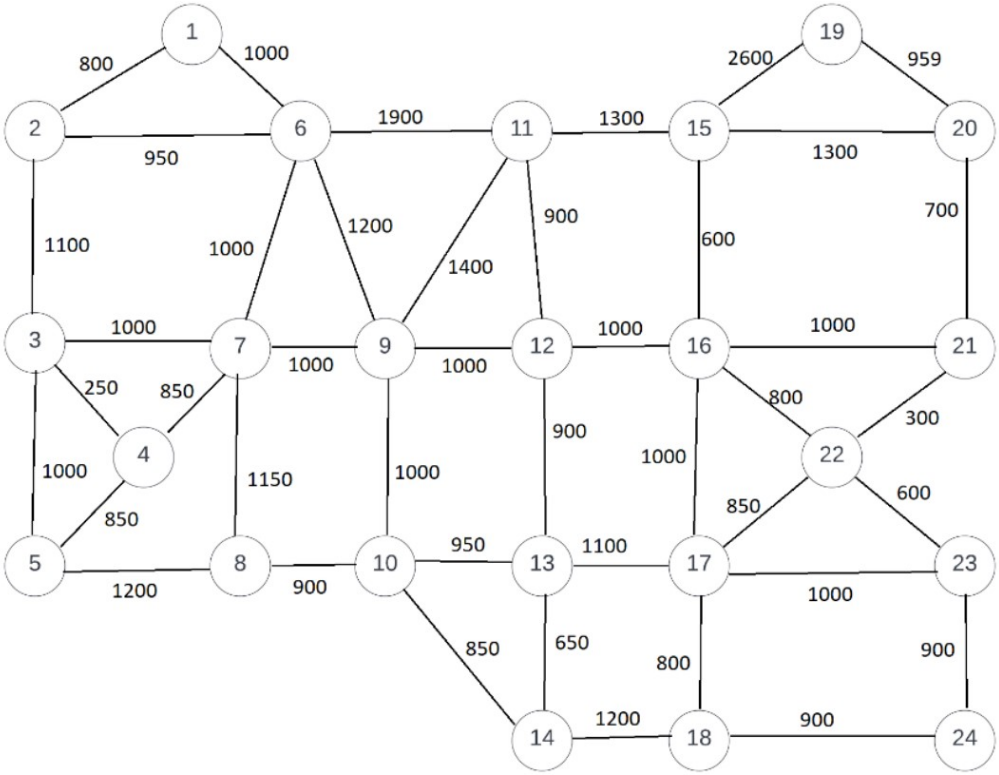


Figure B.5: UBN.

B.2 Bar Charts with the Link Loads for Constrained Routing Test

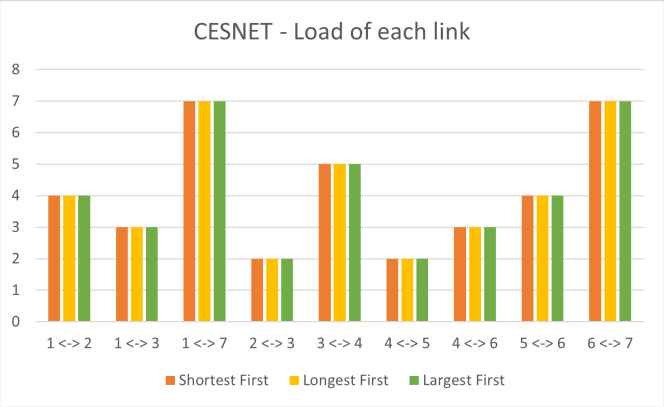


Figure B.6: Bidirectional Link loads of the CESNET for link capacity of 75.

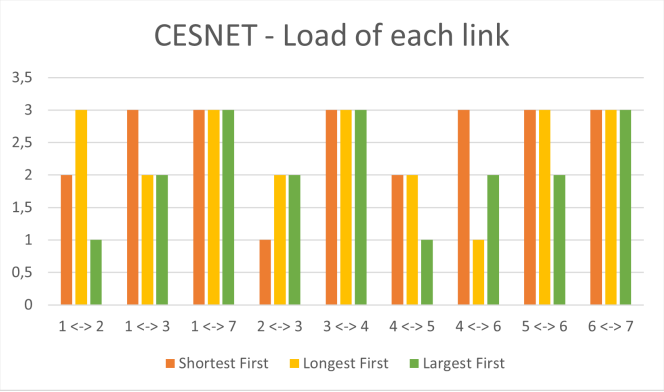


Figure B.7: Bidirectional Link loads of the CESNET for link capacity of 3.

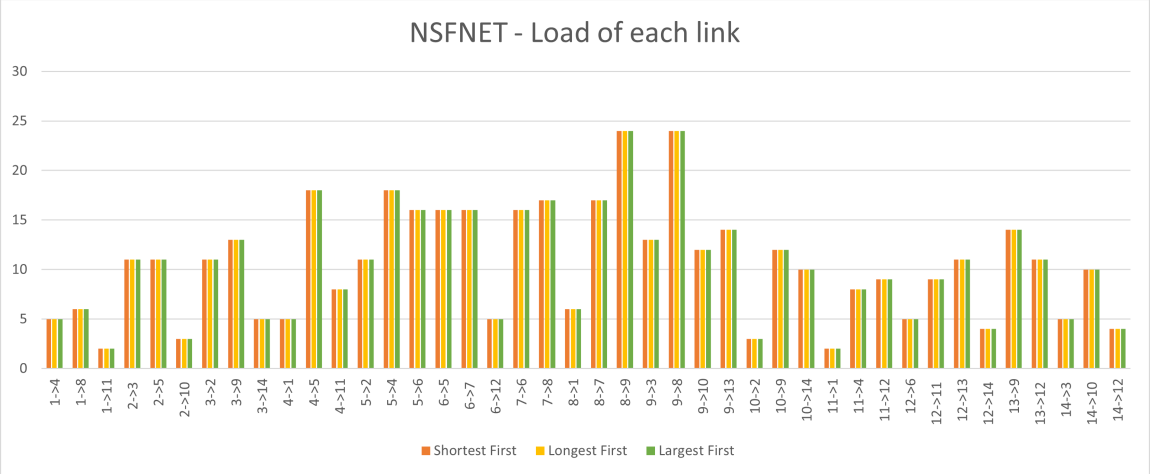


Figure B.8: Link loads of the NSFNET for link capacity of 75.

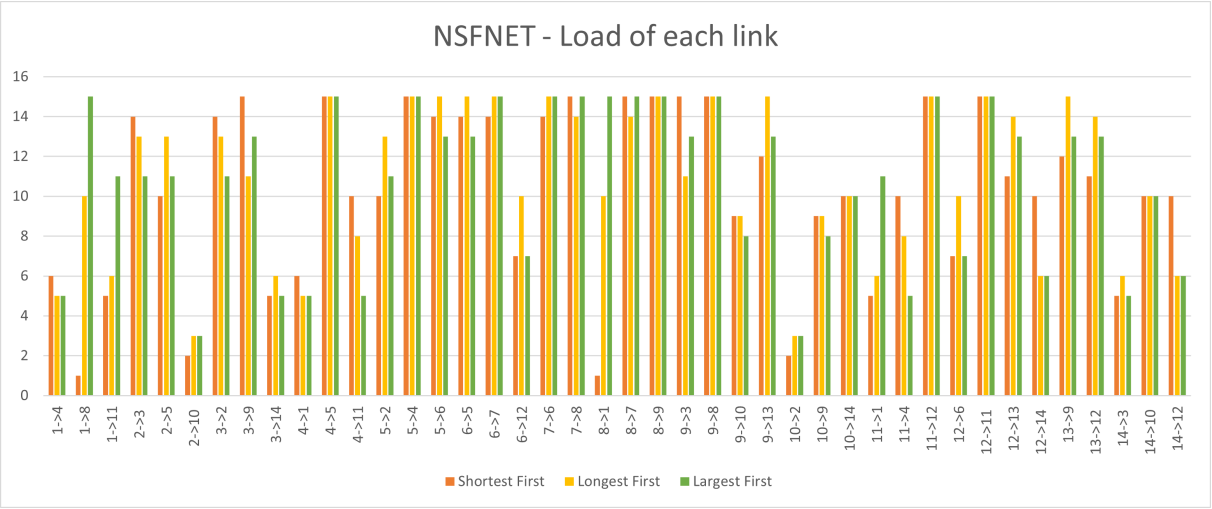


Figure B.9: Link loads of the NSFNET for link capacity of 15.

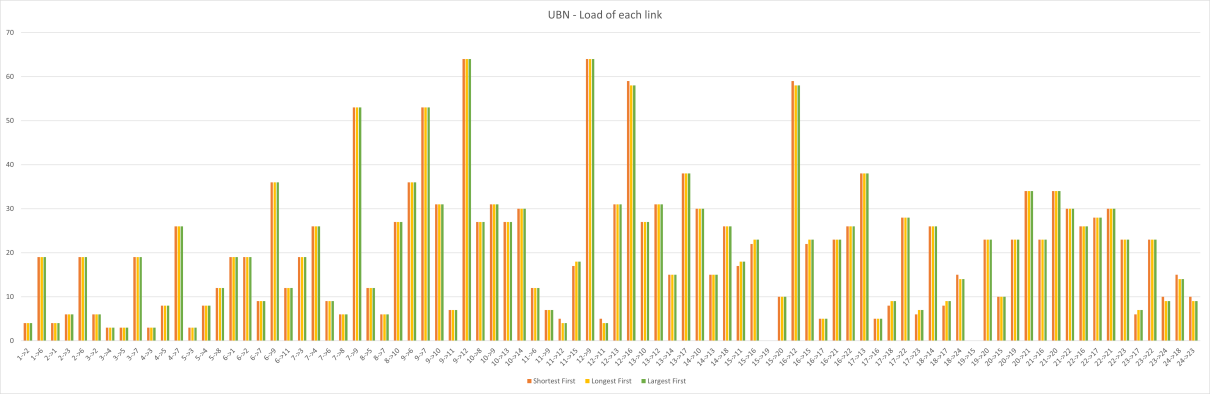


Figure B.10: Link loads of the UBN for link capacity of 75.

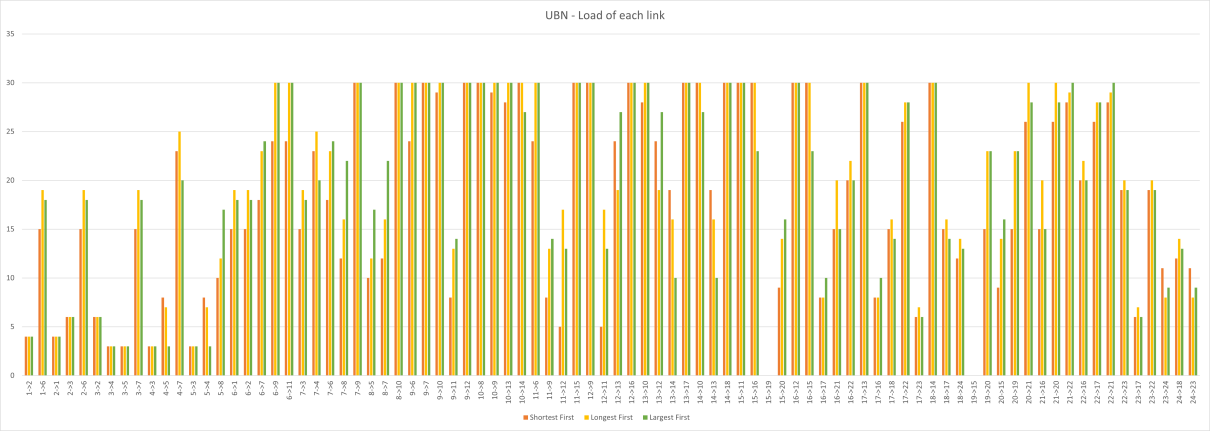


Figure B.11: Link loads of the UBN for link capacity of 30.

B.3 Structure of the DNN Models

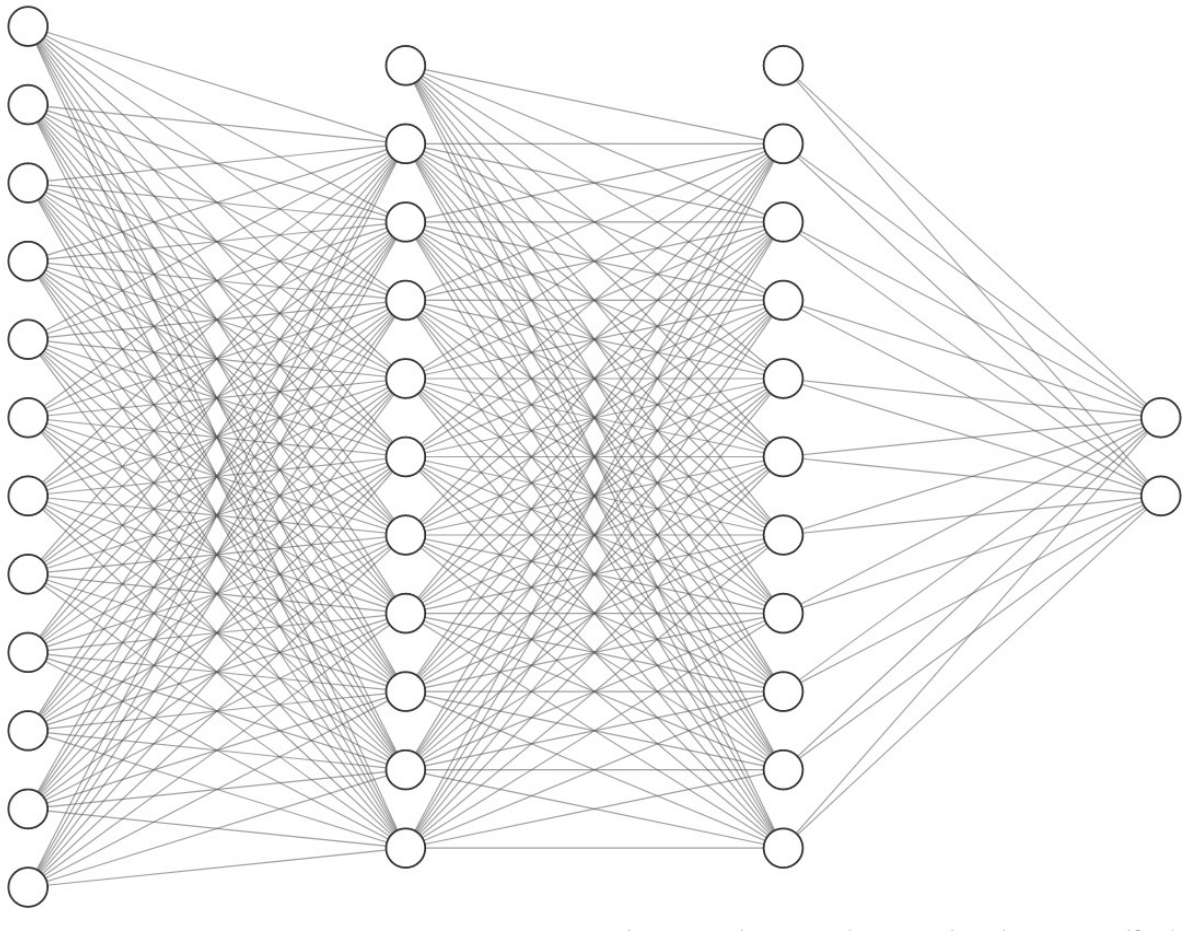


Figure B.12: Structure of the First DNN Model (including the bias terms).

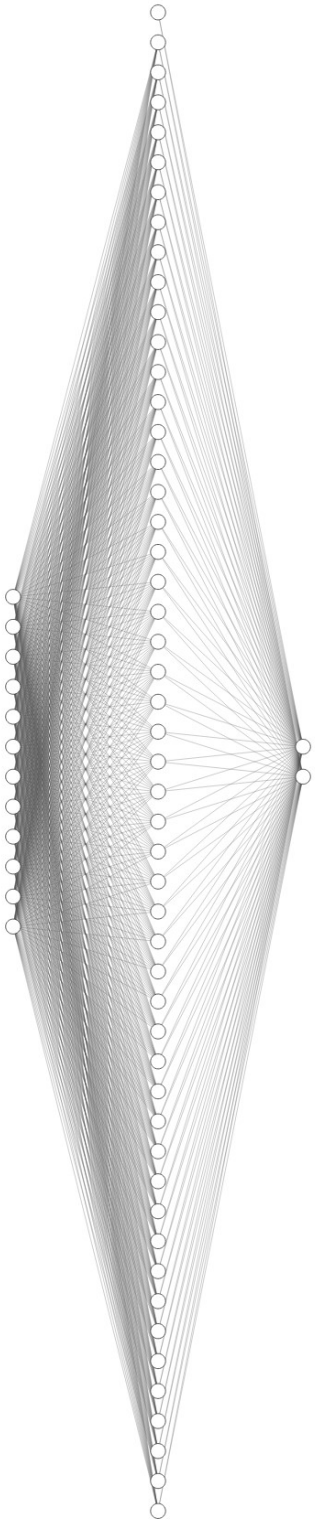


Figure B.13: Structure of the Second DNN Model (including the bias terms).