UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Leveraging graph neural networks for optimization and traffic compression in network digital twins

Felician Paul Almasan Puscas

Leveraging Graph Neural Networks for Optimization and Traffic Compression in Network Digital Twins

Thesis for the Doctoral Degree in Computer Architecture

by

Felician Paul Almasan Puscas

Advisors:
Dr. Albert Cabellos Aparicio
Dr. Pere Barlet Ros

July 2023

Barcelona Neural Networking Center (BNN)
Departament d'Arquitectura de Computadors (DAC)
Universitat Politècnica de Catalunya (UPC)

Acknowledgments

I would like to express my deepest gratitude to my advisors Prof. Albert Cabellos and Prof. Pere Barlet for your guidance and support throughout my PhD journey. Your expertise, encouragement, and determined support have been instrumental in helping me complete my thesis. I am grateful for the opportunities you provided me to present my work and for your support in face of challenges and obstacles. I will always be grateful for the knowledge and experience I have gained under your supervision.

I would like to extend my gratitude to Prof. Krzysztof Rusek for welcoming me at the AGH University of Science and Technology to do a 6 months research stay. Also the knowledge and skills I gained under your supervision have been invaluable, and I cannot thank you enough for sharing them with me. Your hospitality, expertise and technical skills have been an immense source of inspiration, and I am honored to have had the opportunity to work with you.

I would also like to express my deepest appreciation to all the members of Telefónica Research where I spent 3 months developing my skills and collaborating with great researchers. Thanks for giving me this unique opportunity to learn about the mobile networks industry. I am specially grateful to Andra Lutu and Jose Suárez Varela for their valuable guidance and support during my stay at Telefónica Research.

Thanks to all the members and former members of the Barcelona Neural Networking Center and the Computer Architecture Department: Jose Suárez Varela, Jordi Paillissé Vilanova, Sergi Abadal, Ismael Castell, Albert López, Miquel Ferriol Galmés, Guillermo Bernárdez, Carlos Güemes Palau, David Pujol Perich, Berta Serracanta, Hamid Latif, Axel Wassington and Kurdman Rasol. Your kindness, generosity, and camaraderie have made my PhD journey a truly memorable experience.

This thesis has been partially funded by the Spanish I+D+i project TRAINER-A (ref. PID2020-118011GB-C21), funded by MCIN/ AEI/10.13039/501100011033. This work is also partially funded by the Secretariat for Universities and Research of the Ministry of Business and Knowledge of the Government of Catalonia and the European Social Fund. I am very grateful to them for making this PhD possible and for funding my research stay at AGH University of Science and Technology, in Krákow.

I would like to show my gratitude to all my family and friends for your love and encouragement throughout my PhD. Your support has been a source of comfort and strength during the challenging times, and I am truly grateful to have you all in my life. Special thanks to Magda. I could not have accomplished this milestone without your support and I am deeply grateful for all that you have done for me. It was a pleasure to have someone to share and debate complex machine learning topics and new ideas.

Finalmente, me gustaría agradecer el inmenso apoyo y la motivación recibidos de mis padres. Sin ellos no hubiese llegado tan lejos.

Abstract

In recent years, several industry sectors have adapted the Digital Twin (DT) paradigm to improve the performance of physical systems. This paradigm consists of leveraging computational methods to build high-fidelity virtual representations of a physical system or entity. The virtual replica accurately simulates or models the behavior of the physical system without altering its behavior in the real world. Since its inception, the DT has attracted the interest of both academia and industry which can be observed by the growing number of publications, processes, standards and concepts.

The networking community has adapted the DT paradigm with the objective of achieving efficient control and management in modern communication networks. In this context, the Network Digital Twin (NDT) is a renovated concept of classical network modeling tools whose goal is to build accurate data-driven network models. NDTs can be applied to many fundamental networking applications. For example, the NDT allows network operators to design novel network optimization solutions, to perform troubleshooting, what-if analysis, or to plan network upgrades taking into account the network's expected user growth. Since the interaction between the network operator with the NDT does not require access to the real-world network, the aforementioned processes can be carried out in real-time, without jeopardizing the physical network.

This dissertation aims to develop new efficient real-time optimization mechanisms leveraging NDTs. Existing network optimization techniques can be generally divided among optimizer-based solutions (e.g., CP, ILP), heuristics and Machine Learning-based (ML) solutions. Optimizer-based solutions are computationally intensive and they suffer from scalability issues where the optimization time and the problem instance size scale at different speeds. The methods based on heuristics are solutions designed by human experts, making strong assumptions and simplifications on the original problem to reduce its complexity and to make the problem tractable by humans. This is a lengthy process that makes solutions be far-from-optimal, achieving poor network performance at a high cost for the network operator. Finally, existing ML-based solutions need to re-train the ML model every time there is a change in the optimization scenario (e.g., link failure). However, training ML models is a costly process which impedes the application of such methods on real-time network optimization.

The first part of this dissertation proposes an optimization architecture that integrates Graph Neural Networks (GNN) into Deep Reinforcement Learning (DRL). This architecture leverages the planning strategies of DRL and the generalization capabilities of GNNs to optimize over arbitrary network topologies in real-time without the need of re-training the DRL agent. The experimental results show that the DRL+GNN architecture is robust to

operate in real-world topologies that largely differ from the scenarios seen during training. In our work, we evaluate the proposed architecture on two real-world network optimization scenarios. The first scenario is in optical transport networks and the second scenario is in IP networks.

Training DRL agents and building NDTs requires storing large datasets that include a wide range of network states and configurations. However, the network size has been growing both in traffic volume and number of connected devices. Boosted by the deployment of 5G networks and the adaptation of new industry paradigms (e.g., Internet of things), the growing trend is expected to continue for several years. Consequently, storing such large volumes of network-related information can be challenging. The second part of this thesis proposes a new data compression method based on GNNs capable of exploiting spatial and temporal correlations naturally present in network traffic traces, outperforming widely used compression methods such as GZIP.

Resum

En els últims anys, diversos sectors industrials han adaptat el paradigma del "Digital Twin" (DT) per millorar el rendiment dels sistemes físics. Aquest paradigma consisteix en aprofitar mètodes computacionals per construir representacions virtuals d'alta fidelitat d'un sistema o entitat física. La rèplica virtual simula o modela amb precisió el comportament del sistema físic sense alterar el seu comportament en el món real. Des de la seva creació, el DT ha suscitat l'interès tant de l'acadèmia com de la indústria, el que es pot observar pel creixent nombre de publicacions, processos, estàndards i conceptes.

La comunitat de xarxes ha adaptat el paradigma del DT amb l'objectiu d'aconseguir un control i una gestió eficients en les xarxes de comunicació modernes. En aquest context, el "Network Digital Twin" (NDT) és un concepte renovat de les eines de modelatge de xarxes clàssiques que té com a objectiu construir models de xarxes precisos basats en dades. Els NDT es poden aplicar a moltes aplicacions fonamentals de xarxes. Per exemple, els NDT permeten als operadors de xarxes dissenyar noves solucions d'optimització de xarxes, realitzar resolució de problemes, anàlisi de supòsits o planificar actualitzacions de xarxa tenint en compte el creixement esperat dels usuaris de la xarxa. A més, els processos esmentats es poden dur a terme en temps real sense posar en perill la xarxa física.

Aquesta dissertació té com a objectiu desenvolupar nous mecanismes d'optimització en temps real eficients aprofitant els NDTs. Les tècniques d'optimització de xarxa existents es poden dividir en solucions basades en optimitzadors, solucions basades en heurístiques i solucions basades en aprenentatge automàtic (ML). Les solucions basades en optimitzadors són intensives computacionalment i pateixen problemes d'escalabilitat, on el temps d'optimització i la mida de la instància del problema escalen a diferents velocitats. Els mètodes basats en heurístiques són solucions dissenyades per experts humans, un procés molt costós que aconsegueix baixos nivells de rendiment. Finalment, les solucions basades en ML existents necessiten reentrenar el model ML cada vegada que hi ha un canvi en l'escenari d'optimització. No obstant això, el procés de formació dels models ML és costós i dificulta l'aplicació d'aquests mètodes en l'optimització de xarxes en temps real.

La primera part d'aquesta dissertació proposa una arquitectura d'optimització que integra les xarxes neuronals basades en grafs (GNN) en el reforçament profund de l'aprenentatge (DRL). Aquesta arquitectura aprofita les estratègies de planificació del DRL i les capacitats de generalització de les GNN per optimitzar topologies de xarxa arbitràries en temps real sense necessitat de reentrenar l'agent DRL. Els resultats experimentals demostren que l'arquitectura DRL+GNN és robusta per operar en topologies del món real que difereixen considerablement dels escenaris vistos durant la formació. En el nostre treball, avaluem la proposta d'arquitectura en dos escenaris d'optimització de xarxes del món real.

L'entrenament d'agents DRL i la construcció de NDTs requereixen l'emmagatzematge de grans conjunts de dades que inclouen una àmplia gamma d'estats i configuracions de xarxa. No obstant això, la mida de la xarxa ha anat creixent tant en volum de tràfic com en nombre de dispositius connectats. Impulsat per la implementació de les xarxes 5G i l'adaptació de nous paradigmes industrials (per exemple, Internet de les coses), es preveu que la tendència de creixement continuï durant diversos anys. En conseqüència, l'emmagatzematge d'aquestes grans quantitats d'informació relacionada amb la xarxa pot ser un repte. La segona part d'aquesta tesi proposa un nou mètode de compressió de dades basat en les GNN capaç d'aprofitar les correlacions espacials i temporals presents de forma natural en les traces de tràfic de la xarxa, superant els mètodes de compressió àmpliament utilitzats com GZIP.

Resumen

En los últimos años, diversos sectores industriales han adaptado el paradigma del "Digital Twin" (DT) para mejorar el rendimiento de los sistemas físicos. Este paradigma consiste en aprovechar métodos computacionales para construir representaciones virtuales de alta fidelidad de un sistema o entidad física. La réplica virtual simula o modela con precisión el comportamiento del sistema físico sin alterar su comportamiento en el mundo real. Desde su creación, el DT ha despertado el interés tanto de la academia como de la industria, lo que se puede observar por el creciente número de publicaciones, procesos, estándares y conceptos.

La comunidad de redes ha adaptado el paradigma del DT con el objetivo de lograr un control y una gestión eficientes en las redes de comunicación modernas. En este contexto, el "Network Digital Twin" (NDT) es un concepto renovado de las herramientas de modelado de redes clásicas que tiene como objetivo construir modelos de redes precisos basados en datos. Los NDT se pueden aplicar a muchas aplicaciones fundamentales de redes. Por ejemplo, los NDT permiten a los operadores de redes diseñar nuevas soluciones de optimización de redes, realizar resolución de problemas o planificar actualizaciones de red. Además, los procesos mencionados se pueden llevar a cabo en tiempo real sin poner en peligro la red física.

Esta disertación tiene como objetivo desarrollar nuevos mecanismos de optimización en tiempo real eficientes aprovechando los NDTs. Las técnicas de optimización de red existentes se pueden dividir en soluciones basadas en optimizadores, soluciones basadas en heurísticas y soluciones basadas en aprendizaje automático (ML). Las soluciones basadas en optimizadores son intensivas computacionalmente y sufren problemas de escalabilidad, donde el tiempo de optimización y el tamaño de la instancia del problema escalan a diferentes velocidades. Los métodos basados en heurísticas son soluciones diseñadas por expertos humanos, un proceso muy costoso que logra bajos niveles de rendimiento. Finalmente, las soluciones basadas en ML existentes necesitan reentrenar el modelo ML cada vez que hay un cambio en el escenario de optimización. No obstante, el proceso de formación de los modelos ML es costoso y dificulta la aplicación de estos métodos en la optimización de redes en tiempo real.

La primera parte de esta disertación propone una arquitectura de optimización que integra las redes neuronales basadas en grafos (GNN) en el refuerzo profundo del aprendizaje (DRL). Esta arquitectura aprovecha las estrategias de planificación del DRL y las capacidades de generalización de las GNN para optimizar topologías de red arbitrarias en tiempo real sin necesidad de reentrenar el agente DRL. Los resultados experimentales demuestran que la arquitectura DRL+GNN es robusta para operar en topologías del mundo real que

difieren considerablemente de los escenarios vistos durante la formación. En nuestro trabajo evaluamos la propuesta de arquitectura en dos escenarios de optimización de redes del mundo real.

El entrenamiento de agentes DRL y la construcción de NDTs requieren el almacenamiento de grandes conjuntos de datos que incluyen una amplia gama de estados y configuraciones de red. Sin embargo, el tamaño de la red ha ido creciendo tanto en volumen de tráfico como en número de dispositivos conectados. Impulsado por la implementación de las redes 5G y la adaptación de nuevos paradigmas industriales (por ejemplo, Internet de las cosas), se prevé que la tendencia de crecimiento continúe durante varios años. En consecuencia, el almacenamiento de estas grandes cantidades de información relacionada con la red puede ser un desafío. La segunda parte de esta tesis propone un nuevo método de compresión de datos basado en las GNN capaz de aprovechar las correlaciones espaciales y temporales presentes de forma natural en las trazas de tráfico de la red, superando los métodos de compresión ampliamente utilizados como GZIP.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, the simultaneous digital transformation of both society and industry lead to the emergence of a novel set of network applications. These applications have complex requirements that cannot be easily met by traditional network management solutions (e.g., network over-provisioning, admission control). For example, novel forms of communication (e.g., AR/VR or holographic telepresence) require ultra-low deterministic latency, while recent industrial developments (e.g., vehicular networks) need to adapt to highly dynamic networks in real-time. At the same time, the number of connected devices has been growing rapidly, making modern networks' behavior highly dynamic and heterogeneous [2–4]. Consequently, communication networks have become a pillar of today's society, but they are also more complex and costly to manage. This puts pressure on Internet Service Providers (ISP) to ensure customer's quality of service and to fulfill service-level agreements. Therefore, ISPs are challenged to efficiently and effectively manage their network infrastructure to guarantee previously agreed network performance thresholds for different network users and applications.

Boosted by new industry paradigms such as Internet of Things (IoT) or vehicular communications, the expected trend for the following years is that the number of connected devices and the traffic volume will keep increasing. As an example, 5G networks are currently being deployed in many countries, naturally increasing the number of connected devices and the traffic volume. Simultaneously, the research community is already investigating 6G networks, which are expected to go beyond connecting people and include sensors, vehicles, robots and computing resources, among others [5,6]. New industry paradigms such as Industry 4.0 are going to benefit from interconnecting factories and machinery [7]. In addition, there are ongoing projects that are tailoring to extend existing communication networks towards outside of the planet earth. One example is the Starlink project from SpaceX that leverages satellite connectivity to increase the coverage of internet access in remote locations (e.g., deserts, mountains). Another example is the collaboration between NOKIA and NASA to build the first LTE network on the moon [8]. The deployment of

heterogeneous applications come with a wide range of stringent network requirements (e.g., ultra low deterministic latency), increasing the complexity to manage modern communication networks. Consequently, network operators need to find new methods to operate modern heterogeneous networks efficiently.

In the last years, the idea of programmable networks emerged to flexibilize the network management process. Specifically, the Software Defined Network (SDN) paradigm proposed to decouple the network control decisions (control plane) from the underlying forwarding devices (data plane) [9, 10]. In this context, the control plane is in charge of establishing how the packets are being forwarded to reach the destination (e.g., creates the routing table). The role of the data plane is to actually forward the packets. The controller changes the data plane behavior using well-established application programming interfaces such as OpenFlow [11]. In addition, SDN offers a centralized overview of the network state, enabling network operators to easily change network's behavior using software and without having to configure the forwarding devices individually. By breaking down the network management in two planes, SDN facilitates the implementation of new abstractions and the deployment of new methods to manage the network.

While computer networks were living the *software* revolution, a series of breakthroughs in Machine Learning (ML) triggered the beginning of a new ML era. In 2012 a Convolutional Neural Network (CNN) achieved a top-5 error of 15.3% [12] in the Imagenet challenge [13]. This was a novel architecture designed to process images and the resulting error was more than 10.8% points lower than the second best solution that year. In 2016, DeepMind created AlphaGo [14], a ML system based on Neural Networks (NN) that won the best human world player in the game of Go. In 2018, researchers from Google developed BERT, a language model with a transformer-based architecture that achieved outstanding performance in several Natural Language Processing (NLP) tasks. More recently, in 2020 a transformer-based model called AlphaFold achieved outstanding performance in the Critical Assessment of protein Structure Prediction challenge (a.k.a., CASP) [15]. These are only a few examples where ML models had a large impact, outperforming the state of the art. As a result, the research community started to investigate the use of ML to solve real-world problems. Some examples are estimating the arrival time in Google Maps using NNs [16], making weather forecasts with deep generative models [17] and improving the medical imaging process chain with deep learning [18], among others.

In this context, new networking paradigms emerged as a consequence of the ever increasing network management complexity. It is the case of Knowledge Defined Networking (KDN) [19] where the centralization offered by SDN combined with the latest advances in network analytics and monitoring tools enable the implementation of the Knowledge Plane (KP) originally proposed in [20]. Another emerging paradigm is self-driving or autonomous networks [21, 22], which consists on networks that automatically manage themselves without human intervention. More recently, the Network Digital Twin (NDT)

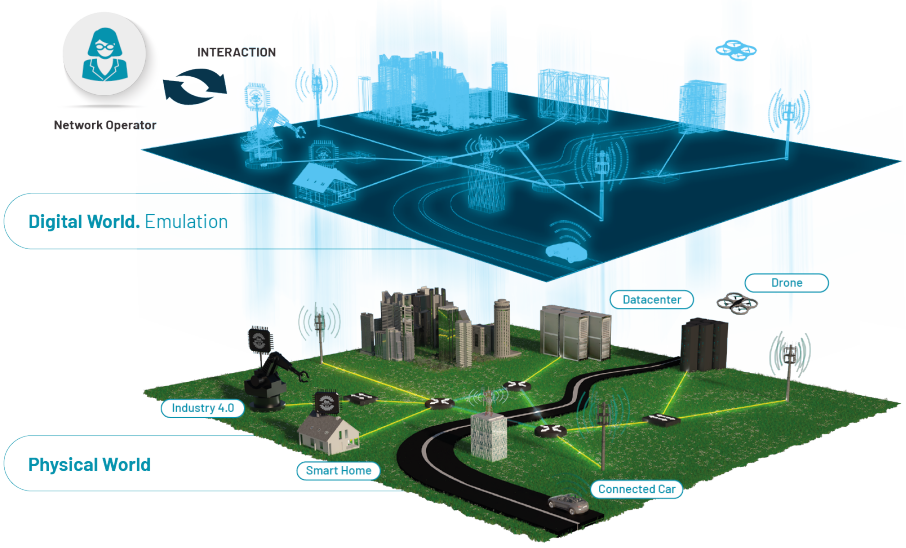


Figure 1.1: A virtual representation of the physical network is built in the digital world. The network operator can interact with the virtual replica to find the best policies to manage the real-world network.

paradigm emerged as a key enabler for efficient control and management of modern communication networks [1, 23]. This dissertation is developed in the context of leveraging NDTs for efficient network management.

A Digital Twin (DT) can be understood as a virtual model of a physical object, system, or phenomenon that is represented in the digital world. The main advantage of DTs is that they can accurately model complex systems at a smaller cost than other methods (e.g., simulation). Nowadays, real-world DT applications include enabling smart manufacturing in Industry 4.0, improving the performance of complex engineering products (e.g., engine design) or modeling physical interactions (e.g., gravitational systems). In a networking context, a NDT is a digital representation of the physical network or event built in the digital world. Figure 1.1 shows a graphical representation of the NDT paradigm. The network operator can interact with the NDT in real-time with the goal of optimizing some desired network performance metrics (e.g., find the routing configuration that minimizes the average delay).

The NDT paradigm aims to achieve accurate data-driven network models that can be used to efficiently model and operate communication networks in real-time. In this vein, the use of ML enables training network models directly with real network data, avoiding the strong assumptions of analytical models (e.g., queueing theory). ML models can thus help achieve similar accuracy to traditional computationally-expensive modeling tools (e.g., packet-level simulation) while keeping a limited execution cost similar to lightweight analytical models. This allows network operators to accurately control the network at much

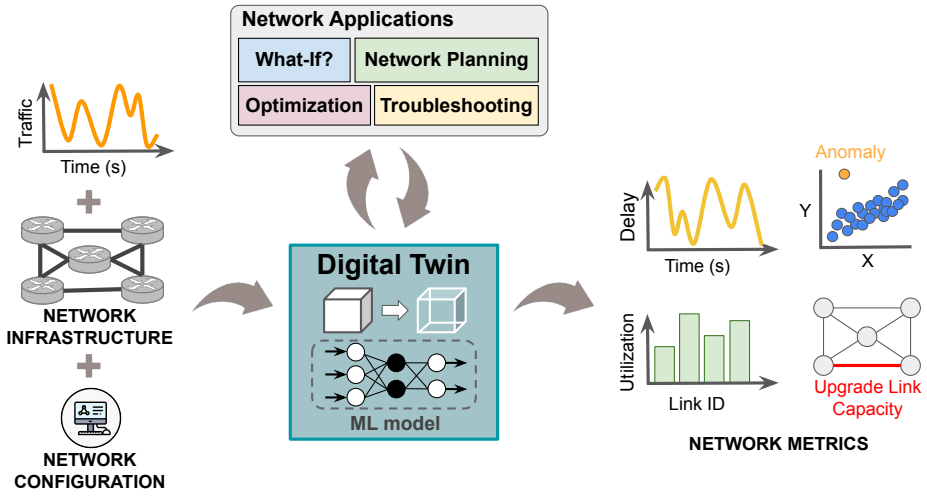


Figure 1.2: General overview of the network digital twin architecture [1].

shorter timescales, enabling real-time operation. There is a growing interest from several communities in building NDTs. In particular, the research community has already proposed several architectures and methods to implement different components of the NDT [23–25]. In addition, standards development organizations (SDO), such as the IETF or the ITU, have already started to work on the definition of a NDT [26, 27].

The central component of the NDT architecture is the DT, which implements a network model using ML. This model is built using network state-related information (e.g., traffic, topology, routing, scheduling policies) and it outputs some estimated network-related metrics (e.g., utilization, delay, anomalies). The outputs of the NDT can be of multiple types depending on the network application (e.g., time series, link-level predictions, global network-level metrics). Figure 1.2 presents the reference architecture of the NDT paradigm [1]. Once the NDT is built, several network applications can leverage it to reach their predefined goals by interacting in real-time. To train these ML-based NDTs, data from real-world networks, dedicated network testbeds, or network simulation tools can be used. This data should be diverse enough to cover a wide representation of potential scenarios that the network operator wants to mimic (e.g., various congestion levels, link failures).

Since the NDT is a faithful copy of the real-world network, the network operator can test any input values, even if these values might cause service disruptions. This is because the DT is executed in a safe environment isolated from the real-world network. Consequently, the network operator can test multiple network configurations to understand their impact on the actual physical network. This improves the solution space search and it enables achieving better network configurations that meet the network operator's goals. In addition, the NDT offers fast inference times, enabling real-time network optimization. Note that the example depicted in Fig. 1.2 illustrates the case of a NDT applied to a fixed

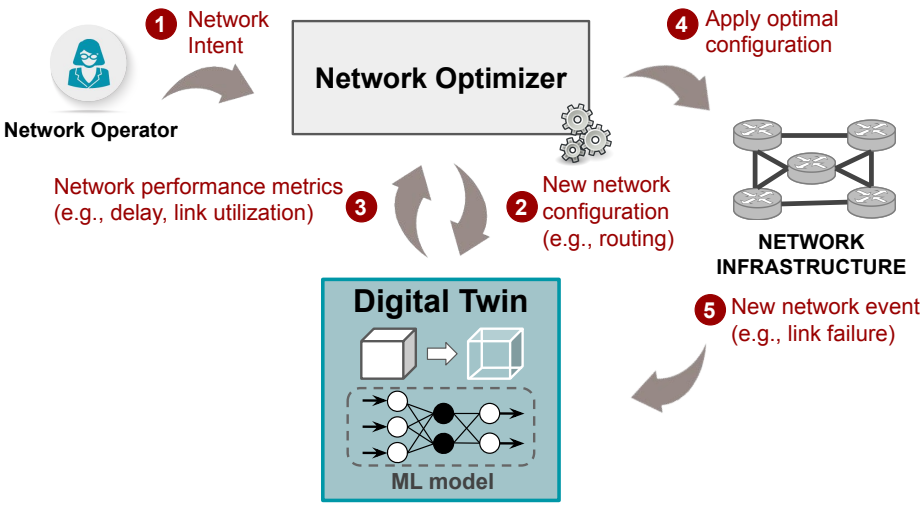


Figure 1.3: Network optimization process with the NDT [1].

network, while analogous architectures could be applied to other kinds of networks, such as wireless or mobile networks.

1.1 Motivation and Objectives

A NDT can be used to design novel optimization solutions for efficient network management. Specifically, the network operator can build a NDT of its physical network and apply meaningful tests on it (e.g., remove links to simulate link failures, test different network configurations). Then, the NDT would indicate what is the network's performance under the new network configurations obtained by the network optimizer. In other words, the network operator can "simulate" worst-case network scenarios and have an estimation of network's performance on these solutions at a very low computational cost, as opposed to the use of network simulators (e.g., OMNet++ [28]). In addition, the NDT can also be leveraged to improve the design of optimization solutions or to detect critical design flaws that might cause service disruptions.

The NDT can be combined with a network optimizer to solve network optimization problems (e.g., routing optimization). In particular, network optimizers can use the NDT to obtain immediate network performance estimations during an optimization process. Figure 1.3 summarizes this process. In the first step, network operators use a declarative language to define the network requirements (e.g., minimize average delay). Then, the optimizer is in charge of searching for the best network configuration that fulfills the predefined requirements (step 2). When the network optimizer sends a network configuration to the DT, this returns the expected network performance if the configuration would be applied in the real-world network. If the performance metrics from the DT indicate that the solution is not good enough (step 3), then the network optimizer continues the search by trying a

new network configuration. The search continues until a stopping condition is met (e.g., number of iterations, average delay below some threshold). Lastly, the best solution found can then be applied to the real network (step 4). Notice that the optimization process can be implemented as a closed-loop, with no human intervention required.

The objective of this dissertation is to develop new efficient real-time optimization mechanisms leveraging NDTs using Deep Reinforcement Learning (DRL) and Graph Neural Networks (GNN). Specifically, we wanted to design a novel architecture that integrates GNNs into DRL to implement the *Network Optimizer* from Figure 1.3. To test the capabilities of the DRL+GNN architecture, we evaluated it on two real-world optimization scenarios where we work with the network traffic, network topology, routing configuration and the per-link utilizations. In both optimization scenarios, the NDT was implemented with Python as the network behavior for both optimization problems can be easily simulated with code by simply adding the traffic crossing each link (see Section 3.2.1 and Section 4.2.1). Notice that in more sophisticated optimization problems (e.g., minimize the average end-to-end delay), the NDT would be implemented with sophisticated ML models that mimic complex behaviors such as network delay.

1.2 Contributions and Outline of the Thesis

The first contribution of this dissertation is a DRL+GNN architecture for network optimization with generalization capabilities evaluated on two optimization scenarios. The first scenario corresponds to optimization in Optical Transport Networks (OTN). The experimental results validated the DRL+GNN architecture and showed us promising results for real-time routing optimization, outperforming widely used heuristics and SoA DRL-based methods. Afterwards, we wanted to improve this architecture and evaluate it in more complex and realistic scenarios. Therefore, we designed a novel DRL+GNN method to solve routing optimization in IP networks. The evaluation results show that the new method can achieve close-to-optimal performance in less than 30 seconds for a set of arbitrary real-world network topologies not seen during the training process.

Storing network traffic information (e.g., packet traces, link-level traffic measurements, flow-level measurements) is important to train the DRL+GNN architecture on real-world data. For example, the traffic evolution from a real-world network could be stored and used to train the DRL agent. In addition, the NDT paradigm also requires the storage and analysis of vast amounts of network traffic data [1]. This is because the ML models will learn to mimic the physical network's behavior from the collected data. The more data we have, better will the the optimization performance of the DRL agent and the higher will be the accuracy of the ML models that implement the NDT. However, storing this kind of data requires an extremely large amount of storage capacity. For example, in the context of mobile networks, one month of traffic traces of ≈9,600 base stations from the city

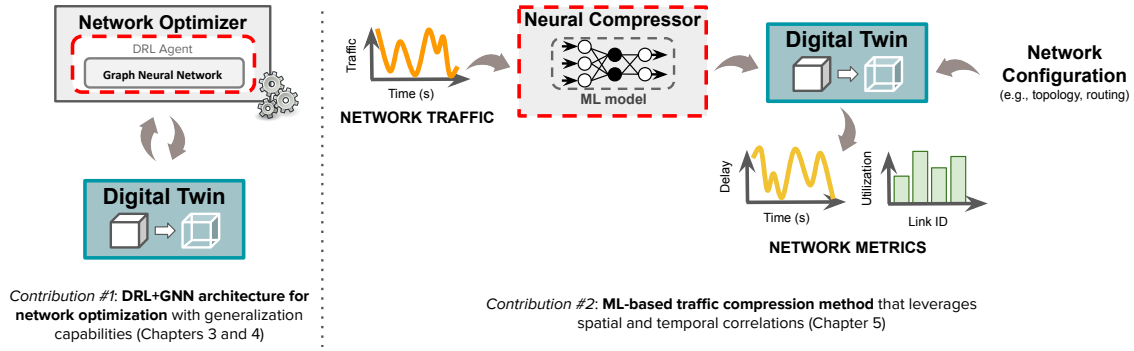


Figure 1.4: Overview of the contributions of this dissertation marked in red. The DRL+GNN architecture is proposed in Section 3 and further improved in Section 4. The traffic compression method is described in Section 5.

of Shanghai takes around 2,4 petabytes of data [29]. This becomes even more challenging when storing data from a mobile network covering an entire country.

Traditionally, network traffic traces are compressed using GZIP [30], a popular lossless method for compressing files regardless of their format (e.g., text, csv files, PDFs). Network operators typically collect traffic traces in PCAP format [31] and they simply compress them with GZIP or similar tools. However, GZIP is a generic compression tool, resulting in sub-optimal compression performance when applied to network traffic data.

The second contribution of this dissertation is a new GNN-based compression method that compresses link-level traffic measurements. Past works showed that network traffic traces are far from being purely random, meaning that they intrinsically have some underlying structure [32–36]. In particular, traffic traces are known to present spatial and temporal patterns that could potentially be exploited to increase current compression ratios. Therefore, we wanted to understand if recent advancements in NN architectures could effectively be used to leverage such correlations to achieve better compression ratios than traditional tools such as GZIP. Figure 1.4 shows an overview of the contributions of this dissertation.

In the following Background section (Section 2) we provide context about the state of the art and some basics on the different technologies that were used in our contributions. The rest of the thesis is structured as follows:

Section 3 : Optimization in Optical Networks

In this section we propose a novel architecture based on DRL and GNN technologies for network control and optimization in OTN. We chose this scenario for its simplicity, which is required to understand the benefits of the DRL+GNN architecture. In our work we directly optimize over the network topology, which can be seen as a graph where the nodes represent routers. Specifically, we propose to integrate DRL with GNNs to build an

agent that correctly routes network traffic demands and maximizes the network's resources usage.

Section 4 : Optimization in IP Networks

In this section, we improve the architecture presented previously and we apply it to a more complex and realistic scenario in IP networks. Specifically, the new DRL+GNN architecture is used to reconfigure an initial routing configuration based on shortest path routing, trying to minimize the utilization of the most congested link. To mitigate the uncertainty naturally intrinsic in ML-based models we integrated a heuristic based on local search to improve the solution of the DRL+GNN architecture at a small overhead.

Section 5 : Network Traffic Compression

In this chapter we present a new compression method that effectively compresses link-level network traffic traces. The proposed method is based on GNNs that leverage spatial and temporal correlations naturally present in network traffic to effectively compress traffic traces.

Section 6 : Related Work

This section offers an overview of the related work. First, we describe the work related to the network optimization scenarios. Then, we discuss the existing work related to network traffic compression. In both cases, we discuss solutions that are based on ML and those that are based on traditional optimization methods (e.g., CP, ILP), outlining the main differences and limitations for each of them.

Section 7 : Conclusions and Future Work

The final section of this dissertation concludes our work and summarizes different research lines as future work.

Chapter 2

Background

2.1 State of the Art

There are multiple optimization techniques that could be used to implement a network optimizer that efficiently manages the network infrastructures. The main objective of the optimizer would be to steer the traffic to achieve a certain goal, such as minimizing the utilization of the most congested link. Network optimization problems can be formulated as Integer Linear Programming (ILP) problems and they can be solved using state-of-the-art (SoA) optimizer engines such as Gurobi [37] or CPLEX [38]. Solutions based on ILP have the benefit of achieving optimal solutions to optimization problems if executed long enough [39–41]. However, when the problem size grows (i.e., the number of nodes and links grows), the number of decision variables increases and the solution space becomes larger and more complex to explore. Consequently, ILP solvers could take several hours or days to find the exact solutions in real-world problems as they have in the order of hundreds of links and nodes [42, 43].

An alternative is the use of Constraint Programming (CP) [44] methods. CP defines the combinatorial problem to solve with a set of decision variables (e.g., traffic demands, OSPF weights), a set of domains (i.e., potential values of the decision variables) and a set of constraints on the feasible solutions (e.g., maximum link utilization must be below a threshold). They have the limitation of being computationally intensive, resulting in sub-optimal solutions if the execution time is not long enough. Computer networks experience external events frequently (e.g., link failures, increase in traffic demand) [45–47], altering the normal network behavior. This forces the CP solver to start the optimization problem from scratch every time there is a network event, impeding efficient real-time network optimization [42, 48].

Network operators can also build a network optimizer using heuristics or expert knowledge [49, 50]. However, the network size and traffic have been growing by almost doubling every year [2–4]. A new set of stringent network requirements came together with this

Method	Execution cost	Performance	Examples
Heuristic	Low	Low	[49, 50]
Mathematical Optimizers (e.g., CP, ILP)	High	High	[39–42, 48]
SoA Machine Learning	High (training)	High	[55–58]

Table 2.1: Summary of the SoA to implement a *Network Optimizer*.

growth (e.g., ultra low deterministic latency), raising the complexity of efficient real-time network operation. As a result, the design of high performance heuristics for network optimization became more challenging for humans, and with a higher cost for network operators.

ML-based solutions showed high performance in real-world optimization problems [14, 51–54]. As a result, the networking community started investigating the use of ML for solving network optimization problems [55–58]. However, existing ML-based solutions fail to generalize when applied to different network scenarios. Generalization refers to the ability of the ML model to adapt to new network scenarios not seen during training (e.g., new network topologies, new configurations). As a result, these methods require re-training the ML model when there is a change in the network optimization problem, increasing the execution cost for modern networks which are highly dynamic [2–4]. Table 2.1 shows an overview of the existing SoA methods for network optimization.

In this dissertation we propose a DRL+GNN architecture that can generalize to other scenarios not seen during training. With generalization, the DRL agent does not need to be retrained every time there is a change in the optimization problem (e.g., link failure). This means that we effectively reduce the high training cost from SoA ML solutions, enabling real-time optimization. In addition, GNNs enable achieving high performance because they were specifically designed to learn from graph-structured information. As computer networks are graphs, we argue that GNNs are a key technology for network optimization.

2.2 Deep Reinforcement Learning

DRL algorithms aim at learning a long-term strategy that maximizes an objective function in an optimization problem. Typically, DRL agents start from a *tabula rasa* state and they learn the optimal strategy by an iterative process that explores the state and action spaces. These are denoted by a set of states ($\mathcal{S}$) and a set of actions ($\mathcal{A}$). Given a state $s \in \mathcal{S}$, the agent will perform an action $a \in \mathcal{A}$ that produces a transition to a new state $s' \in \mathcal{S}$, which will provide the agent with a reward r. This reward will indicate to the DRL agent how good the action was performed. Then, the objective is to find a strategy that maximizes the cumulative reward by the end of an episode.

The definition of the end of an episode depends on the optimization problem to address. As an example, in the game of chess the end of an episode is when one of the two players has won the game or when the game is in stalemate. In other scenarios, the end of an episode

States \ Actions	a0	a1	a2	. . .	aN
s0	Q(s0,a0)	Q(s0,a1)	Q(s0,a2)		Q(s0,a2)
s1	Q(s1,a1)	Q(s1,a1)	Q(s1,a2)		Q(s1,a2)
s2	Q(s2,a1)	Q(s2,a1)	Q(s2,a2)		Q(s2,a2)
. . .				. . .	. . .
sN	Q(sN,a1)	Q(sN,a1)	Q(sN,a2)	. . .	Q(sN,a2)

Figure 2.1: Q-learning table with all the possible actions and states. Initially, the table is initialized with zeros or random values. During training, the table values are updated using the Bellman equation.

can be defined by a maximum number of steps performed by the DRL agent. There is no better way to define the end of an episode but it is of extreme relevance to design a reward that will lead the agent towards the optimization end goal.

2.2.1 DQN

Q-learning [59] is a model-free reinforcement learning algorithm whose goal is to make an agent learn a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$. The algorithm creates a table (a.k.a., q-table) with all the possible combinations of states and actions. This table is exemplified in Figure 2.1 and it indicates the maximum future expected reward of taking an action in a specific state.

At the beginning of training, the table is initialized with zeros or random values. During training, an agent that interacts with the environment updates these values according to the rewards obtained after selecting an action. These values, called q-values, represent the expected cumulative reward after applying action a in state s, assuming that the agent follows the current policy π during the rest of the episode. During training, the table values or q-values are updated using the Bellman equation (see Equation 2.2.1) where $Q(s_t,a_t)$ is the q-value function at time-step t, α is the learning rate, $r(s_t,a_t)$ is the reward obtained from selecting action a_t from state s_t and $\gamma \in [0,1]$ is the discount factor.

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha\left(r(s_t,a_t) + \gamma \max_{a'} Q(s'_t,a') - Q(s_t,a_t)\right) \tag{2.2.1}$$

Deep Q-Network (DQN) [60] is a more advanced algorithm based on Q-learning that uses a Deep Neural Network (DNN) to approximate the q-value function. As the q-table size becomes larger, Q-learning faces difficulties to learn a policy from high dimensional state and action spaces. To overcome this problem, they proposed to use a DNN as a q-value function estimator, relying on the generalization capabilities of DNNs to estimate

Algorithm 1 PPO Pseudocode

Input : actor policy parameters θ_1, critic value function parameters ϕ_1
1: **for** e = 1,2, ... to MAX_NUM_EPISODES **do**
2: Collect trajectories D_e by interacting with the environment using policy $\pi(\theta_e)$
3: Compute rewards-to-go R_e
4: Compute advantage estimates A_e using any advantage estimation method
5: Update the parameters θ_{e+1} using A_e, maximizing the loss with gradient ascent
6: Fit the critic value function using R_e, computing the mean-squared error and gradient descent

the q-values of states and actions unseen in advance. For this reason, a DNN well suited to understand and generalize over the input data of the DRL agent is crucial for the agents to perform well when facing states (or environments) never seen before. Additionally, DQN uses an experience replay buffer to store past sequential experiences (i.e., stores tuples of $\{s,a,r,s'\}$). While training the neural network, the temporal correlation is broken by sampling randomly from the experience replay buffer.

2.2.2 PPO

Another family of RL algorithms is the policy optimization based methods. The difference with value-based methods (e.g., DQN) is that they try to optimize the policy directly. Specifically, the agent's policy π_θ (a|s) is parameterized by θ and the reward that indicates how good the action performed by the DRL agent depends on the π_θ function. Policy optimization methods use gradient ascent to find the best parameters θ that produce the highest rewards.

There are various algorithms that can be applied to find the best parameters θ. In our work, we used the Proximal Policy Optimization (PPO) for its high performance in complex control tasks [61]. PPO is a policy gradient learning algorithm that utilizes the actor-critic method to train the policy network π_θ (a|s) [62]. This function can be seen as a neural network. This method consists of training two networks: the actor and the critic. The actor is in charge of mapping a state observation to an action and the critic estimates the expected reward of the agent for the given action. Another key difference between DQN and PPO is that the latter learns in an online fashion, without storing the past experiences in a large replay buffer. Particularly, PPO stores a batch of experiences (e.g., an entire episode where the agent interacts with the environment), performs the gradient update and discards the batch. Algorithm 1 shows the pseudocode of the PPO algorithm. We refer the reader to the original paper [61] for a more detailed description.

PPO generally works better than DQN because it converges faster to a better policy in complex scenarios. In addition, PPO can be easily applied to problems with continuous action space, while DQN needs to pass through an action discretization process. Conversely, PPO suffers from having a high variance when estimating the gradients, which could neg-

atively impact the convergence of the algorithm. There are several techniques which can be used to minimize the variance, such as TD-lambda [63] and Generalized Advantage Estimation [64], which were both included in our PPO implementation.

2.3 Graph Neural Networks

The key technology behind the contributions of this dissertation are the GNNs. Thes are a novel family of neural networks designed to operate over graphs. They were introduced in [65] and numerous variants have been developed since [66, 67]. In their basic form, they consist of associating some initial states to the different elements of an input graph, and combining them considering how these elements are connected in the graph. An iterative algorithm updates the elements' state and uses the resulting states to produce an output. The particularities of the problem to solve will determine which GNN variant is more suitable, depending on, for instance, the nature of the graph elements (i.e., nodes and edges) involved.

Message Passing Neural Networks (MPNN) [68] are a well-known type of GNNs that apply an iterative message-passing algorithm to propagate information between the nodes of the graph. In a message-passing step, each node k receives messages from all the nodes in its neighborhood, denoted by *N(k)*. Messages are generated by applying a message function *m(·)* to the hidden states of node pairs in the graph. Then, they are combined by an aggregation function, for instance, a sum (Equation 2.3.1). Finally, an update function *u(·)* is used to compute a new hidden state for every node (Equation 2.3.2).

$$M_k^{t+1} = \sum_{i \in N(k)} m(h_k^t, h_i^t) \tag{2.3.1}$$

$$h_k^{t+1} = u(h_k^t, M_k^{t+1}) \tag{2.3.2}$$

Functions *m(·)* and *u(·)* can be learned by neural networks. After a certain number of iterations, the final node states are used by a readout function *r(·)* to produce an output for the given task. This function can also be implemented by a neural network and is typically tasked to predict properties of individual nodes (e.g., the node's class) or global properties of the graph.

GNNs have been able to achieve relevant performance results in multiple domains where data is typically structured as a graph [68, 69]. Since computer networks are fundamentally represented as graphs, it is inherent in their design that GNNs offer unique advantages for network modeling compared to traditional neural network architectures (e.g., fully connected NN, Convolutional NN, etc.).

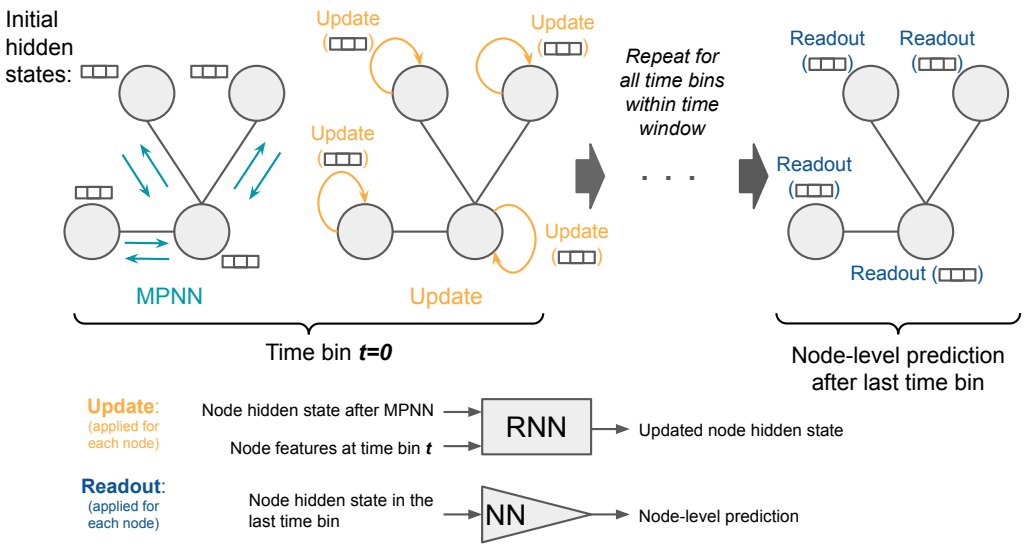


Figure 2.2: The ST-GNN uses a time window to indicate how long is the sequence to process before making the final prediction. For each time bin the MPNN learns the spatial correlations and then the RNN updates the node-level hidden states. Once the entire sequence is processed, a readout function is in charge of making the final predictions with the resulting node-level hidden states. Notice that the predictions can be global (i.e., aggregating all hidden states) or per-node using for example a multilayer perceptron NN architecture.

2.3.1 Spatio-Temporal GNN

There are kinds of data that in addition to spatial dependencies can also present time-varying characteristics. As an example, consider the following scenario in transportation systems where the nodes of a graph represent the cities of a country and the edges correspond to the roads connecting these cities. Due to external factors, spatial relationships among cities will change during time. For example, traffic congestion, accidents or adverse weather could make it difficult or impede reaching some cities using some specific roads at a given time.

Spatio-Temporal Graph Neural Networks (ST-GNN) are a different neural architecture designed to learn spatial correlations and temporal dependencies stored as a time-varying graph. In general, ST-GNN architectures consist on a GNN for spatial modeling (i.e., dependencies between graph edges/nodes) and a Recurrent Neural Network (RNN) to learn the temporal trends [70–72]. The ST-GNN uses a time window to indicate how long is the temporal sequence to process before making the prediction. For each time bin within the window, the GNN and the RNN are alternated, updating the graph entities' hidden states sequentially.

In this dissertation we built a ST-GNN composed of a MPNN and a RNN for learning the spatial and temporal correlations respectively. Figure 2.2 shows an overview of the ST-GNN and how the MPNN and RNN are alternated for each time bin. Initially, all nodes'

hidden states are initialized with the node-level features and we do not consider the edge entity. Then, a MPNN is executed using the initialized node hidden states to learn the spatial relations between nodes. Afterwards, the RNN takes the resulting hidden states for the corresponding time bin and updates each node independently. This final hidden state is used to initialize the node features in the next time bin, repeating the same process until the end of the time window. Finally, after having iterated over all time bins within the window, a final NN is applied for each node to make node-level predictions. In the case of graph-level prediction, the node-level final hidden states could be aggregated (e.g., sum, mean) and a NN could process the resulting graph-level hidden state to make global predictions.

Chapter 3

Optimization in Optical Networks

3.1 Introduction

The first contribution of this dissertation is a DRL+GNN architecture for network optimization with generalization capabilities. In this chapter we describe the architecture and evaluate it by optimizing the routing of traffic demands in an Optical Transport Network (OTN). We chose this scenario because we wanted to evaluate the proposed architecture in a simple scenario to understand its advantages and limitations. We implemented the DT of the optical network using Python (see Figure 1.3). This is because the network behavior for the optimization problem can be easily simulated with code by simply adding the traffic crossing each link in the topology. More sophisticated optimization problems (e.g., minimize the average end-to-end delay) would require for the NDT to be implemented with sophisticated ML models that mimic complex behaviors such as network delay.

DRL has shown significant improvements in sequential decision-making and automated control problems [14, 73]. As a result, the network community started to investigate DRL as a key technology for network optimization [19, 21, 22, 74]. However, existing DRL-based solutions still fail to *generalize* when applied to different network scenarios [55, 58, 75]. In this context, generalization refers to the ability of the DRL agent to adapt to new network scenarios not seen during training (e.g., new network topologies, routing configurations, scheduling policies).

We argue that generalization is an essential property for the successful adoption of DRL technologies in production networks. Without generalization, DRL solutions should be trained in the same network where they are deployed, which is not possible or affordable in general. To train a DRL agent is a costly and lengthy process that often requires significant computing power and the instrumentation of the network to observe its performance (e.g., delay, jitter). Additionally, decisions made by a DRL agent during training can lead to degraded performance or even to service disruption. Thus, training a DRL agent in the customer's network may be unfeasible.

With generalization, a DRL agent can be trained with multiple, representative network topologies and configurations. Afterwards, it can be applied to other topologies and configurations, as long as they share some common properties. Such a "universal" model can be trained in a laboratory and later on be incorporated in a product or a network device (e.g., router, load balancer). The resulting solution would be ready to be deployed to a production network without requiring any further training or instrumentation in the customer network[1].

Existing DRL proposals for networking were designed to operate in the same network topology seen during training [55, 58, 75], thereby limiting their potential deployment on production networks. The main reason behind this strong limitation is that computer networks are fundamentally represented as graphs. For instance, the network topology and routing policy are typically represented as such. However, SoA proposals [55, 56, 76, 77] use traditional NN architectures (e.g., fully connected, convolutional) that are not well suited to model graph-structured information [78].

GNNs [65] were proposed to model and operate over graphs with the aim of achieving relational reasoning and combinatorial generalization. In other words, GNNs facilitate learning the relations between graph elements and the rules for composing them. GNNs have shown unprecedented generalization capabilities in the field of network modeling and optimization [79].

In our work, we integrate GNNs into DRL agents to solve network optimization problems. Particularly, we propose an architecture that is intended to solve routing optimization in optical networks and to generalize over never-seen arbitrary topologies. The GNN integrated in our DRL agent is inspired by Message-Passing Neural Networks (MPNN), which were successfully applied to solve a relevant chemistry-related problem [68]. In our work, the GNN was specifically designed to capture meaningful information about the relations between the links and the traffic flowing through the network topologies. Then, our DRL+GNN agent is tasked to allocate traffic demands as they arrive, maximizing the traffic volume routed through the network.

The evaluation results show that the proposed DRL+GNN architecture achieves strong generalization capabilities compared to SoA DRL (SoA DRL) algorithms. This is important as it indicates that our solution does not need to be retrained when there are changes in the network topology (e.g., link failures). Notice that existing methods based on CP or ILP are computationally intensive and they need to restart the optimization process from scratch when there is a change in the optimization problem (e.g., link failure). In addition, SoA DRL-based solutions need to re-train the DRL agent every time there is a change in the optimization problem (e.g., traffic changes, link failure). This is a computationally intensive process that impedes real-time network optimization.

[1]Note that solutions based on transfer learning do not offer this property as DRL agents need to be re-trained on the network where they finally operate.

Overall, our DRL+GNN architecture for network optimization has the following features:

- *Generality:* It can work effectively in network topologies and scenarios never seen during training.
- *Deployability:* It can be deployed to production networks without requiring training nor instrumentation in the customer network.
- *Low overhead:* Once trained, the DRL agent can make routing decisions in only one step ($\approx ms$), while its cost scales linearly with the network size.
- *Commercialization:* Network vendors can easily embed it in network devices or products, and successfully operate "arbitrary" networks.

We believe the combination of these features can enable the development of a new generation of networking solutions based on DRL that are more cost-effective than current approaches based on heuristics or linear optimization. All the topologies and scripts used in the experiments, as well as the source code of our DRL+GNN agent are publicly available [80].

3.2 Background

The solution proposed in our work combines two machine learning mechanisms. First, we use a GNN to model computer network scenarios. GNNs are neural architectures specifically designed to generalize over graph-structured data [78], and thus, are well suited to operate successfully in other network scenarios including topologies and routing configurations unseen during training. In addition, they offer near real-time operation in the scale of milliseconds (see Section 3.5.2). Second, we use DRL to build an agent that learns how to efficiently operate networks following a particular optimization goal. DRL applies the knowledge obtained in past optimizations to later decisions, without the necessity to run computationally intensive algorithms. The combination with GNNs enables the DRL agent to generalize and operate efficiently on scenarios never seen before.

3.2.1 Problem Statement

In this section, we explore the potential of a GNN-based DRL agent to address the routing problem in Optical Transport Networks (OTN). Particularly, we consider a network scenario based on Software-Defined Networking (SDN), where the DRL agent (located in the control plane) has a global view of the current network state, and has to make routing decisions on every traffic demand as it arrives. We consider a traffic demand as the volume of traffic sent from a source to a destination node. This is a relevant optimization scenario

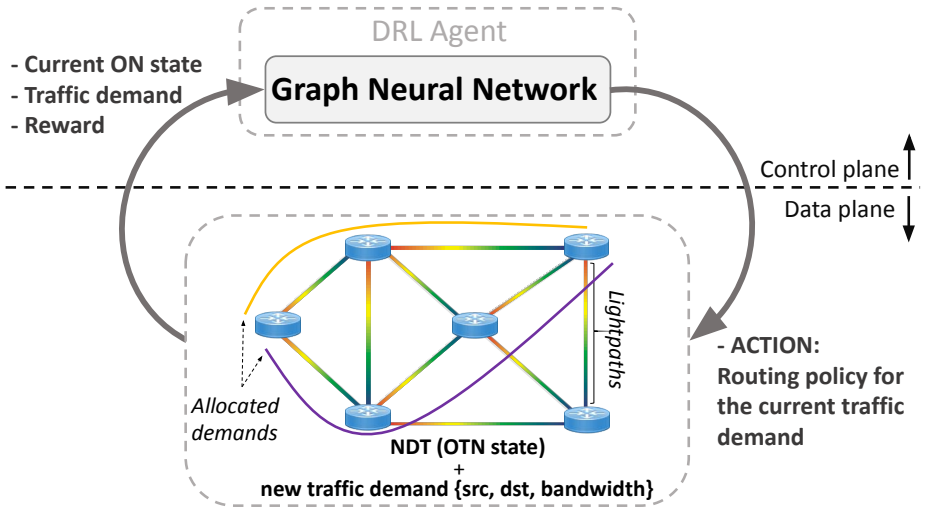


Figure 3.1: Schematic representation of the DRL agent in the OTN routing scenario.

that has been studied in the last decades in the optical networking community, where many solutions have been proposed [55, 56, 81].

In our OTN scenario, the DRL agent makes routing decisions at the electrical domain, over a logical topology where nodes represent Reconfigurable Optical Add-Drop Multiplexers (ROADM) and edges are predefined lightpaths connecting them (see Figure 3.1). The DRL agent receives traffic demands with different bandwidth requirements defined by the tuple $\{src, dst, bandwidth\}$, and it has to select an end-to-end path for every demand. Particularly, end-to-end paths are defined as sequences of lightpaths connecting the source and destination of a demand. Since the agent operates at the electrical domain, traffic demands are defined as requests of Optical Data Units (ODUk), whose bandwidth requirements are defined in the ITU-T Recommendation G.709 [82]. The ODUk signals are then multiplexed into Optical Transport Units (OTUk), which are data frames including Forward Error Correction. Eventually, OTUk frames are mapped to different optical channels within the lightpaths of the topology.

In this scenario, the routing problem is defined as finding the optimal routing policy for each incoming source-destination traffic demand. The learning process is guided by an objective function that aims to maximize the traffic volume allocated in the network in the long-term. We consider that a demand is properly allocated if there is enough available capacity in all the lightpaths forming the end-to-end path selected. Note that lightpaths are the edges in the logical topology where the agent operates. The demands do not expire, occupying the lightpaths until the end of a DRL episode. This implies a challenging task for the agent since it has not only to identify critical resources on networks (e.g., potential bottlenecks), but also to deal with the uncertainty in the generation of future traffic demands. The following constraints summarize the traffic demand routing problem in the OTN scenario:

- The agent must make sequential routing decisions for every incoming traffic demand
- Traffic demands can not be split over multiple paths
- Previous traffic demands can not be rerouted and they occupy the links' capacities until the end of the episode

The *optimal* solution to the OTN optimization problem can be found by solving its Markov Decision Process (MDP) [83]. To do this, we can use techniques such as dynamic programming, which consist of an iterative process over all MDP's states until convergence. The MDP for the traffic demand allocation problem consists of all the possible network topology states and the transition probabilities between states. Notice that in our scenario we have uniform transition probabilities from one state to the next. One limitation of solving MDPs *optimally* is that it becomes infeasible for large and complex optimization problems. As the problem size grows, so does the MDP's state space, where the space complexity (in number of states) is $S \approx O(N^{\mathrm{E}})$, having N as the number of different capacities a link can have and E as the number of links. Therefore, to solve the MDP the algorithm will spend more time on iterating over all MDP's states.

3.3 Proposed Solution

In this section, we describe the proposed DRL+GNN architecture. On the one hand, there is the GNN-based DRL agent which defines the actions to apply on the network state. These actions consist of allocating the demands on one of the candidate paths. The DRL agent implements the DQN algorithm [60], where the q-value function is modeled by a GNN. On the other hand, there is an environment defining the optimization problem to solve. This environment stores the network topology, together with the link features. In addition, the environment is responsible for generating the reward once an action is performed, indicating to the agent if the action was good or not. The environment contains the digital twin of the optical network (see Figure 1.3). The DRL agent interacts with the NDT in order to optimize some network metric. We implemented the NDT using Python code because the network behaviour can be easily simulated by simply adding the traffic crossing each link in the topology.

The learning process is based on an iterative process where at each time step, the agent receives a graph-structured network state observation from the environment. Particularly, the network state includes the topology with some link-level features (e.g., utilization). Then, the GNN constructs a graph representation where the links of the topology are the graph entities. In this representation, the link hidden states are initialized considering the input link-level features and the routing action to evaluate (see more details in Sections 3.3.1, 3.3.2 and 3.3.3). With this representation, an iterative message passing algorithm runs between the links' hidden states according to the graph structure. The output

of this algorithm (i.e., new links hidden states) is aggregated into a global hidden state that encodes topology information, and then is processed by a deep neural network. This process makes the GNN topology invariant because the global hidden state length is pre-defined and it will always have the same length for different topology sizes. At the end of the message passing phase, the GNN outputs a q-value estimate. This q-value is evaluated over a limited set of actions, and finally the DRL agent selects the action with highest q-value.

The selected action is then applied to the environment, producing a transition in the NDT to a new network state. If the action performed was successful (i.e., all the links selected had enough available capacity to support the new demand), a positive reward is returned to the agent, otherwise the episode ends and the DRL agent receives a reward equal to zero. During the training phase, an ϵ-*greedy* exploration strategy [60] is used to select the next action applied by the agent. This means that a random action is executed with probability ϵ, while the action with higher q-value is selected with probability $(1 - \epsilon)$.

3.3.1 Environment

The environment implements the NDT of the optical network and it has a network state defined by the typology's link features, including the link capacities and link betweenness. The former indicates the amount of traffic capacity available on the link. The latter is a measure of centrality inherited from graph theory that indicates how many paths may potentially traverse the link. From the experimental results we observed that this feature helps reduce the grid search of the hyperparameter tuning for the DRL agent. This is because the betweenness helps the agent converge faster to a good policy. In particular, we compute the link betweenness in the following way: for each pair of nodes in the topology, we compute k candidate paths (e.g., the k shortest paths), and we maintain a per-link counter that indicates how many paths pass through the link. Then, the betweenness on each link is the number of end-to-end paths crossing the link divided by the total number of paths.

3.3.2 Action Space

In this section we describe how the routing actions are represented in the DRL+GNN agent. Note that the number of possible routing combinations for each source-destination node pair typically results in a high dimensional action space in large-scale real-world networks. This makes the routing problem complex for the DRL agent, since it should estimate the q-values for all the possible actions to apply (i.e., routing configurations). To overcome this problem, the action space must be carefully designed to reduce the problem dimensionality. In addition, to enable generalization to other topologies, the action space should be equivalent across topologies. In other words, if the actions in the training topology are represented by shortest paths, in the evaluation topology they should also be shortest paths. If the action space would be different (e.g., multiple paths between a source-destination node

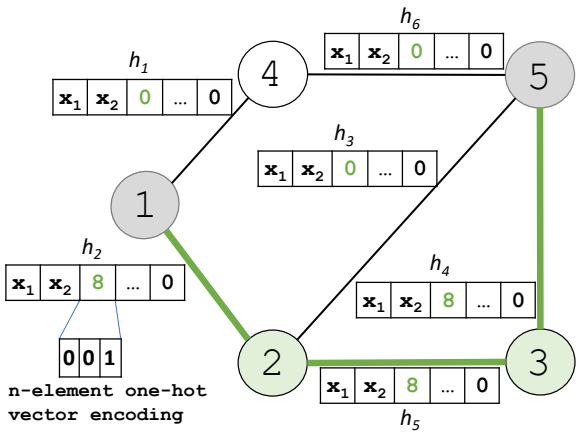


Figure 3.2: Action representation in the link hidden states.

pair), the agent would have problems learning and it would not generalize well. To leverage the generalization capabilities of GNNs, we introduced the action into the agent in the form of a graph. This makes the action representation invariant to node and edge permutation, which means that, once the GNN is successfully trained, it is able to understand actions over arbitrary graph structures (i.e., over different network states and topologies).

Considering the above, we limit the action set to k candidate paths for each source-destination node pair. To maintain a good trade-off between flexibility to route traffic and the cost to evaluate all the possible actions, we selected a set with the k=4 shortest paths (by number of hops) for each source-destination node pair. This follows a criteria originally proposed by [56]. Note that the action set differs depending on the source and destination nodes of the traffic demand to be routed.

To represent the action, we introduce it within the network state. Particularly, we consider an additional link-level feature, which is the bandwidth allocated over the link after applying the routing action. This value corresponds to the bandwidth demand of the current traffic request to be allocated. Likewise, the links that are not included in the path selected by the action will have this feature set to zero. Since our OTN environment has a limited number of traffic requests with various discrete bandwidth demands, we represent

Notation	Description
x_1	Link available capacity
x_2	Link Betweenness
x_3	Action vector (bandwidth allocated)
$x_4 - x_N$	Zero padding

Table 3.1: Input features of the link hidden states. N corresponds to the size of the hidden state vector.

the bandwidth allocated with a N-element one-hot encoding vector, where N is the vector length.

Figure 3.2 illustrates the representation of the action in the hidden state of the links in a simple network scenario. A traffic request from node 1 to node 5, with a traffic demand of 8 bandwidth units, is allocated over the path formed by the nodes $\{1,2,3,5\}$. To summarize, Table 3.1 provides a description of the features included in the links' hidden states. These values represent both the network state and the action, which is the input needed to model the q-value function $Q(s,a)$.

The size of the hidden states is typically larger than the number of features in the hidden states. This is to enable each link to store information of himself (i.e., his own initial features) plus the aggregated information coming from all the links' neighbors (see Section 3.3.3). If the hidden state size is equal to the number of link features, the links won't have space to store information about the neighboring links without losing information. This results in a poor graph embedding after the readout function. On the contrary, if the state size is very large, it can lead to a large GNN model, which can overfit to the data. A common approach is to set the state size larger than the number of features and to fill the vector with zeros.

3.3.3 GNN Architecture

The GNN model is based on the message passing neural network architecture [68]. In our case, we only consider the link entity and we perform the message passing between all links. We choose link entities instead of node entities because the link features are what define the OTN routing optimization problem. Node entities could be added when addressing an optimization problem that needs to incorporate node-level features (e.g., I/O buffer size, scheduling algorithm). Algorithm 2 shows a formal description of the message passing process where the algorithm receives as input the links' features (x_l) and outputs a q-value (q).

Algorithm 2 Message Passing

Input : $\mathbf{x}_l$
Output : $\mathbf{h}_l^T, q$
for each $l \in \mathcal{L}$ **do**
    $h_l^0 \leftarrow [\mathbf{x}_l, 0 \ldots, 0]$
for $t = 1$ to T **do**
    for each $l \in \mathcal{L}$ **do**
        $M_l^{t+1} = \sum_{i \in N(l)} m\left(h_l^t, h_i^t\right)$
        $h_l^{t+1} = u\left(h_l^t, M_l^{t+1}\right)$
$rdt \leftarrow \sum_{l \in \mathcal{L}} h_l$
$q \leftarrow R(rdt)$

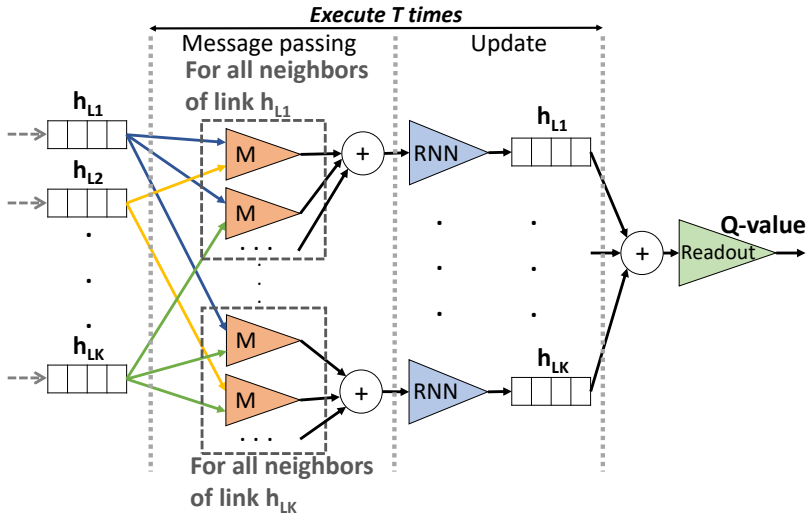


Figure 3.3: Message passing architecture.

The algorithm performs T message passing steps. A graphical representation can be seen in Figure 3.3, where the algorithm iterates over all links of the network topology. For each link, its features are combined with those of the neighboring links using a fully-connected, corresponding to M in Figure 3.3. The outputs of these operations are called *messages* according to the GNN notation. Then, the *messages* computed for each link with their neighbors are aggregated using an element-wise sum (line 5 in Algorithm 2). Afterwards, a Recurrent NN (RNN) is used to update the link hidden states h_{LK} with the new aggregated information (line 6 in Algorithm 2). At the end of the message passing phase, the resulting link states are aggregated using an element-wise sum (line 7 in Algorithm 2). The result is passed through a fully-connected neural network which models the readout function of the GNN. The output of this latter function is the estimated q-value of the input state and action.

The role of the RNN is to learn how the link states change along the message passing phase. As the link information is being spread through the graph, each hidden state will store information from links that are farther and farther apart. Therefore, the concept of time appears. RNNs are a neural architecture that are tailored to capture sequential behavior (e.g., text, video, time-series). In addition, some RNN architectures (e.g., GRU) are designed to process large sequences (e.g., long text sentences in NLP). Specifically, they internally contain gates that are designed to mitigate the vanishing gradients, a common problem with large sequences [84]. This makes RNNs suitable to learn how the links' state evolve during the message passing phase, even for large T.

3.3.4 DRL Agent Operation

The DRL agent operates by interacting with the environment. In Algorithm 3 we can observe a pseudocode describing the DRL agent operation. At the beginning, the environ-

ment *env* initializes all the link features. At the same time, the environment generates a traffic demand to be allocated by the tuple $\{src, dst, bw\}$ and an environment state s. The environment also initializes the cumulative reward to zero, defines the action set size and creates the experience replay buffer (*agt.mem*). Afterwards, a while loop is executed (lines 3-16) that finishes when there is some demand that cannot be allocated in the network topology. For each of the k=4 shortest paths, the demand is allocated along all the links forming the path and the q-value is computed (lines 7-9). Once we have the q-value for each state-action pair, the next action a to apply is selected using an ϵ-greedy exploration strategy (line 10) [60]. The action is then applied to the environment, leading to a new state *s'*, a reward r and a flag *Done* indicating if there is some link without enough capacity to support the demand. Additionally, the environment returns a new traffic demand tuple $\{src', dst', bw'\}$. The information about the state transition is stored in the experience replay buffer (line 13). This information will be used later on to train the GNN in the *agt.replay()* call (line 15), which is executed every M training iterations.

3.4 Experimental Evaluation

In this section we evaluate our GNN-based DRL agent to optimize the routing configuration in the OTN scenario described previously. In particular, the experiments in this section are focused on evaluating the performance and generalization capabilities of the proposed DRL+GNN agent. Afterwards, in Section 3.5, we analyze the scalability properties of our solution and discuss other relevant aspects related to the deployment on production networks.

Algorithm 3 DRL Agent operation

1: $s, src, dst, bw \leftarrow$ *env.init_env()*
2: $reward \leftarrow 0$, $k \leftarrow 4$, $agt.mem \leftarrow \{\ \}$, $Done \leftarrow False$
3: **while** not Done **do**
4: $k\_q\_values \leftarrow \{\ \}$
5: $k\_shortest\_paths \leftarrow$ *compute_k_paths(k, src, dst)*
6: **for** i in $0, ..., k$ **do**
7: $p' \leftarrow$ *get_path(i, k_shortest_paths)*
8: $s' \leftarrow$ *env.alloc_demand(s, p', src, dst, dem)*
9: $k\_q\_values[i] \leftarrow$ *compute_q_value(s', p')*
10: $q\_value \leftarrow$ *epsilon_greedy(k_q_values, ϵ)*
11: $a \leftarrow$ *get_action(q_value, k_shortest_paths, s)*
12: $r, Done, s', src', dst', bw' \leftarrow$ *env.step(s, a)*
13: $agt.rmb(s, src, dst, bw, a, r, s', src', dst', bw')$
14: $reward \leftarrow reward + r$
15: **If** *training_steps % M == 0:* **agt.replay()**
16: $src \leftarrow src'; dst \leftarrow dst'; bw \leftarrow bw', s \leftarrow s'$

3.4.1 Evaluation Setup

We implemented the DRL+GNN solution described in Section 3.3 with Tensorflow [85] and evaluated it on an OTN network simulator implemented using the OpenAI Gym framework [86]. The source code, together with all the training and evaluation results are publicly available [80].

In the OTN simulator, we consider three traffic demand types (ODU2, ODU3, and ODU4), whose bandwidth requirements are expressed in terms of multiples of ODU0 signals (i.e., 8, 32, and 64 ODU0 bandwidth units respectively) [82]. When the DRL agent correctly allocates a demand, it receives an immediate reward being the bandwidth of the current traffic demand if it was properly allocated, otherwise the reward is 0. We consider that a demand is successfully allocated if all the links in the path selected by the DRL agent have enough available capacity to carry such demand. Likewise, episodes end when a traffic demand was not correctly allocated. Traffic demands are generated by uniformly selecting a source-destination node pair and a traffic demand type (ODUk). This makes the problem even more difficult for the DRL agent, since the uniform traffic distribution hinders the exploitation of prediction systems to anticipate possible demands difficult to allocate. In other words, all traffic demands are equally probable to appear in the future, making it more difficult for the DRL agent to estimate the expected future rewards.

Initial experiments were carried out to choose an appropriate gradient-based optimization algorithm and to find the hyperparameter values for the DRL+GNN agent. For the GNN model, we defined the links' hidden states h_l as 27-element vectors (filled with the features described in Table 3.1). Note that the size of the hidden states is related to the amount of information they may potentially encode. Larger network topologies and complex network optimization scenarios might need larger sizes for the hidden state vectors. In every forward propagation of the GNN we execute $T=7$ message passing steps using batches of 32 samples. The optimizer used is the Stochastic Gradient Descent [87] with a learning rate of 10^{-4} and a momentum of 0.9. We start the ϵ-greedy exploration strategy with ϵ=1.0 and maintain this value during 70 training iterations. Afterwards, ϵ decays exponentially every episode. The experience buffer stores 4,000 samples and is implemented as a FIFO queue (first in, first out). We applied *l2* regularization and dropout to the readout function with a coefficient of 0.1 in both cases. The discount factor γ was set to 0.95.

3.4.2 Methodology

We divided the evaluation of our DRL+GNN agent in two sets of experiments. In the first set, we focused on reasoning about the performance and generalization capabilities of our solution. For illustration purposes, we chose two particular network scenarios and analyzed them extensively. As a baseline, we implemented the DRL-based system proposed in [56], a SoA solution for routing optimization in OTNs. Later on, in Section 3.5, we

evaluated our solution on real-world network topologies and analyzed its scalability in terms of computation time and generalization capabilities.

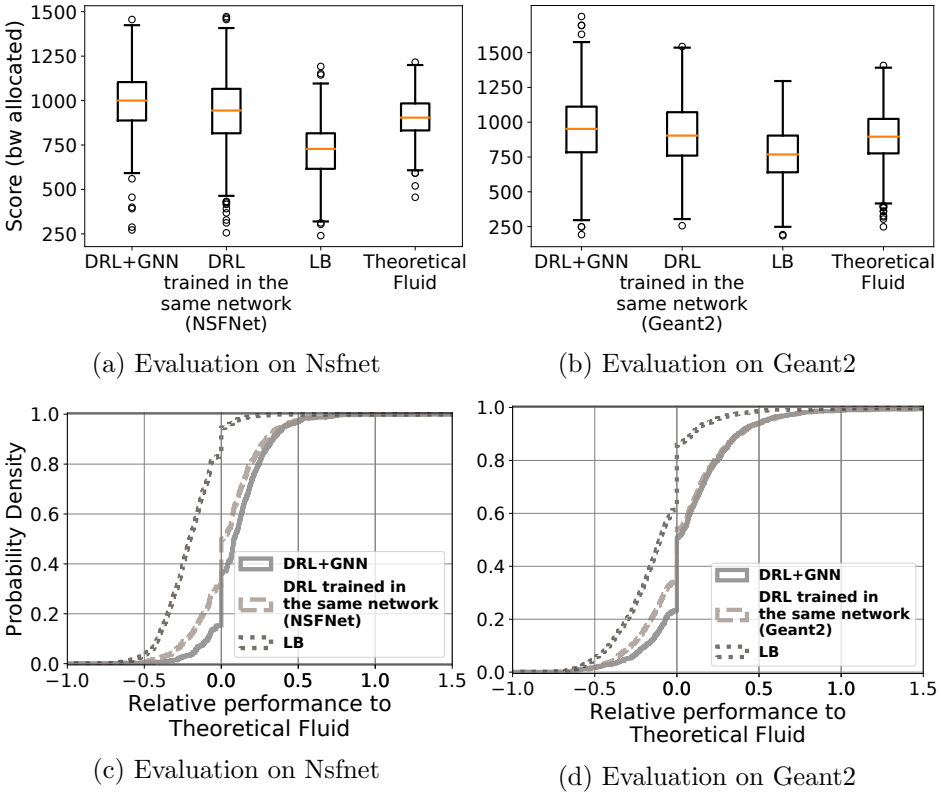
To find the *optimal* MDP solution to the OTN optimization problem is infeasible due to its complexity. Take as an example a small network topology with 6 nodes and 8 edges, where the links have capacities of 3 ODU0 units, there is only one bandwidth type available (1 ODU0) and there are 4 possible actions. The resulting number of states of the MDP is 5^8*6*5*1 $\approx 1.17e7$. To find a solution to the MDP we can use dynamic programming algorithms such as value iteration. However, this algorithm has a time complexity to solve the MDP of $O(S^2A)$, where S and A are the number of states and actions respectively and $S \approx O(N^{\mathrm{E}})$, having N as the number of different capacities a link can have and E as the number of links.

As an alternative, we compare the DRL+GNN agent performance with a theoretical fluid model (labeled as *Theoretical Fluid*). This model is a theoretical approach which considers that traffic demands may be split into the *k=4* candidate paths proportionally to the available capacity they have. This routing policy is aimed at avoiding congestion on links. For instance, paths with low available capacity will carry a small proportion of the traffic volume from new demands. Note that this model is non-realizable because ODU demands cannot be split in real OTN scenarios. However, this model is fast to compute and serves us as a reference to compare the performance of the DRL+GNN agent. In addition, we also use a Load Balancing (LB) routing policy, which selects uniformly random one path among the *k=4* candidate shortest paths to allocate the traffic demand.

We trained the DRL+GNN agent in an OTN routing scenario on the 14-node Nsfnet topology [88], where we considered that the links represent lightpaths with capacity for 200 ODU0 signals. Note that the capacity is shared on both directions of the links and that the bandwidth of different traffic demands is expressed in multiples of ODU0 signals (i.e., 8, 32 or 64 ODU0 bandwidth units). We ran 1,000 training iterations where the agent received traffic demands and allocated them on one of the *k=4* shortest paths available in the action set. The model with highest performance was selected to be benchmarked against traditional routing optimization strategies and SoA DRL-based solutions.

3.4.3 Performance evaluation against state-of-the-art DRL-based solutions

In this evaluation experiment, we compare our DRL+GNN agent against SoA DRL-based solutions. Particularly, we adapted the solution proposed in [56] to operate in scenarios where links share their capacity in both directions. We trained two different instances of the SoA DRL agent in two network scenarios: the 14-node Nsfnet and the 24-node Geant2 topology [89]. We made 1,000 experiments with uniform traffic generation to provide representative results. Note that both, the proposed DRL+GNN agent and the SoA DRL solution, were evaluated over the same list of generated demands.



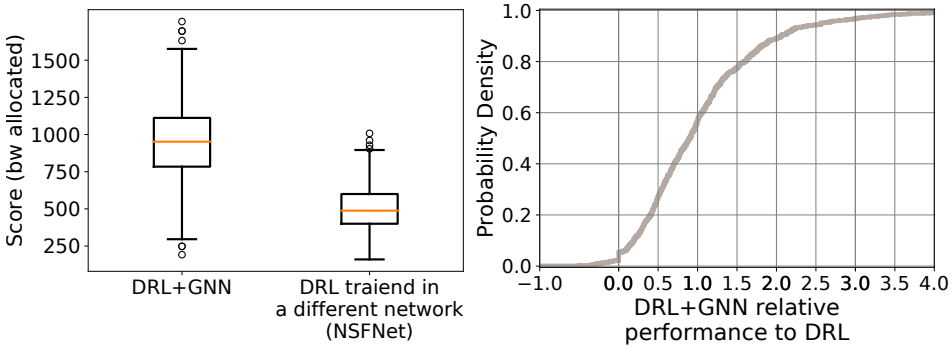
(a) Evaluation on Nsfnet

(b) Evaluation on Geant2

(c) Evaluation on Nsfnet

(d) Evaluation on Geant2

Figure 3.4: Performance evaluation against SoA DRL. Notice that the vertical lines in 3.4c and 3.4d indicate the same performance as the theoretical fluid model.

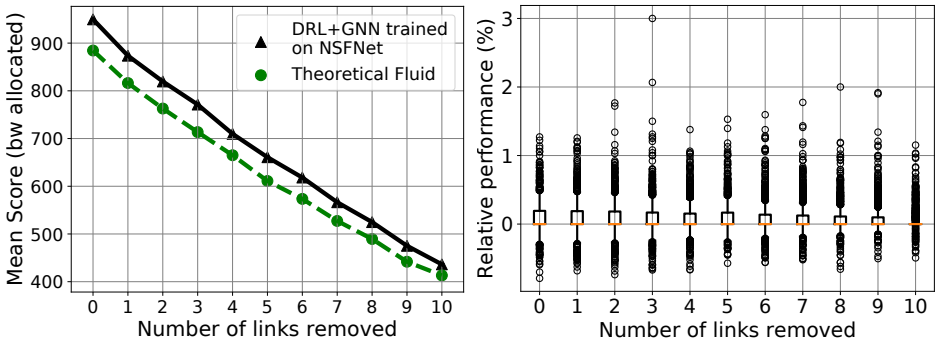


(a) Bandwidth allocated

(b) CDF

Figure 3.5: Evaluation on Geant2 of DRL-based solutions trained on Nsfnet.

We run two experiments to compare the *performance* of our DRL+GNN with the results obtained by the SoA DRL (SoA DRL). In the first experiment, we evaluated the DRL+GNN agent against the SoA DRL agent trained on Nsfnet, the LB routing policy, and the theoretical fluid model. We evaluated the four routing strategies on the Nsfnet topology and compared their performance. In Figure 3.4a, we can observe a boxplot with the evaluation results of 1,000 evaluation experiments. The y-axis indicates the agent score, which corresponds to the bandwidth allocated by the agent. Figure 3.4c shows the



(a) (b)

Figure 3.6: DRL+GNN evaluation on a use case with link failures.

Cumulative Distribution Function (CDF) of the relative score obtained with respect to the fluid model. In this experiment we could also observe that the proposed DRL+GNN agent slightly outperforms the SoA DRL-based by allocating 6.6% more bandwidth. In the second experiment, we evaluated the same models (DRL+GNN, SoA DRL, LB, and Theoretical Fluid) on the Geant2 topology, but in this case the SoA DRL agent was trained on Geant2. The resulting boxplot can be seen in Figure 3.4b and the CDF of the evaluation samples in Figure 3.4d. Similarly, in this case our agent performs slightly better than the SoA DRL approach (3% more bandwidth).

We run another experiment to compare the *generalization capabilities* of our DRL+GNN agent. In this experiment, we evaluated the DRL+GNN agent (trained on Nsfnet) against the SoA DRL agent trained on Nsfnet, and evaluated both agents on the Geant2 topology. The resulting boxplot can be seen in Figure 3.5a and the corresponding CDF in Figure 3.5b. The results indicate that in this scenario the DRL+GNN agent also outperforms the SoA DRL agent. In this case, in 80% of the experiments our DRL+GNN agent achieved more than 45% of performance improvement with respect to the SoA DRL proposal. These results show that while the proposed DRL+GNN agent is able to generalize and achieve outstanding performance in the unseen Geant2 topology (Figure 3.5a and Figure 3.5b), the SoA DRL agent performs poorly when applied to topologies not seen during training. This reveals the lack of generalization capability of the latter DRL-based solution compared to our DRL+GNN agent.

3.4.4 Use case: Link failure resilience

This subsection presents a use case where we evaluate if our DRL+GNN agent can adapt successfully to changes in the network topology. For this, we consider the case of a network with link failures. Previous work showed that real-world network topologies change during time (e.g., due to link failures) [42, 90, 91]. These changes in network connectivity are unpredictable and they have a significant impact in protocol convergence [90] or on fulfilling network optimization goals [42].

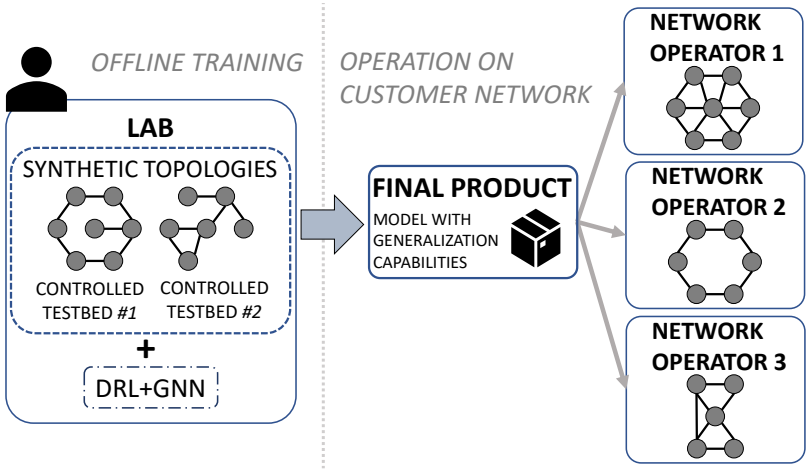


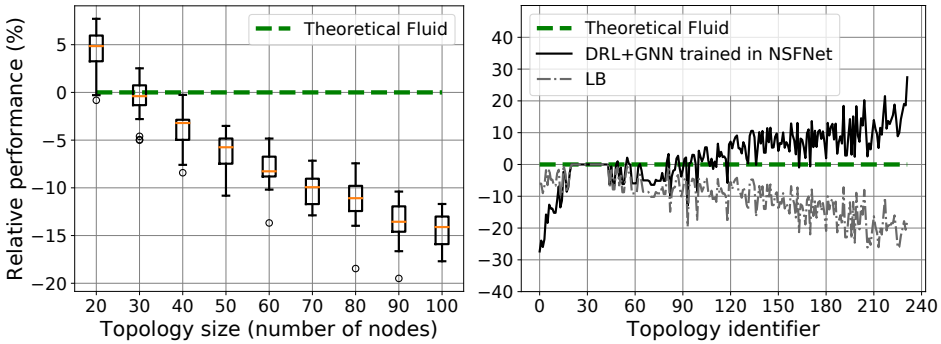
Figure 3.7: DRL+GNN deployment process overview by incorporating it into a product.

In this evaluation, we considered a range of scenarios that can experience up to 10 link failures. Thus, the DRL+GNN agent is tasked to find new routing configurations that avoid the affected links while still maximizing the total bandwidth allocated. We executed experiments where $n \in [1, 10]$ links are randomly removed from the Geant2 topology. We compare the score (i.e., bandwidth allocated) achieved by the DRL+GNN agent with respect to the theoretical fluid model. Figure 3.6a shows the average score over 1,000 experiments (y-axis) as a function of the number of link failures (x-axis). There, we can observe that the DRL+GNN agent can maintain better performance than the theoretical baseline even in the extreme case of 10 concurrent link failures. Likewise, Figure 3.6b shows the relative score of our DRL+GNN agent against the theoretical fluid model. In line with the previous results, the relative score is maintained as links are removed from the topology. This suggests that the proposed DRL+GNN architecture is able to adapt to topology changes.

3.5 Analysis on deployment

In this section we analyze and discuss relevant aspects of the proposed DRL+GNN architecture towards deployment in production networks. In the context of efficient network management, DRL cannot succeed without generalization capabilities. Training a DRL agent requires instrumenting the network with configurations that may disrupt the service. As a result, training in the customer's network may be unfeasible. With generalization capabilities, the DRL agent can be trained in a controlled lab (for instance at the vendor's facilities) and shipped to the customer. Once deployed, it can operate efficiently in an unseen network or scenario. Figure 3.7 illustrates this training and deployment process of a product based on our DRL+GNN architecture.

To better understand the technical feasibility and scalability properties of such a product in terms of cost and generalization, we designed two experiments. First, we analyze how



(a) (b)

Figure 3.8: DRL+GNN relative performance with respect to the fluid model over 180 synthetic topologies (a) and 232 real-world topologies (b).

the effectiveness of our agent scales with the network size, by training it in a single (small) network and evaluating its performance in synthetic and real-world network topologies. Second, we analyze the scalability of our agent in terms of computation time after deployment (i.e., the time it takes for the agent to make routing decisions). This is particularly relevant in real-time networking scenarios. In both scenarios we used the DRL+GNN agent trained in a *single* topology (14 nodes Nsfnet) and we analyzed its performance in larger topologies (up to 100 nodes) not seen during training.

3.5.1 Generalization over network topologies

Synthetic topologies

In this experiment we generated a total of 180 synthetic topologies with an increasing number of nodes. For each topology size –in number of nodes– we generated 20 topologies and we evaluated the agent on 1,000 episodes. To do this, we used the NetworkX python library [92] to generate random network topologies between 20 and 100 nodes with similar *average node degree* to Nsfnet. This allows us to analyze how the network size affects the performance.

Figure 3.8a shows how the performance scales inversely with the topology size. For benchmark purposes, we computed the relative score with respect to the theoretical fluid model. The agent shows a remarkable performance in unseen topologies. As an example, the agent has a similar performance to the theoretical fluid model in the 30-node topologies, which double the size of the single 14-node topology seen during training. In addition, in the 100-node topologies, we observe only a ≈15% drop in performance. This result shows that the generalization properties of our solution degrade gracefully with the size of the network. It is well-known that deep learning models lose generalization capability as the distribution of the data seen during training differs from the evaluation samples (see Section3.5.3). From a deployment standpoint, vendors can always include a wider range of network topologies in the training set to minimize this decrease in performance.

Real-world network topologies

In this section we evaluate the generalization capabilities of our DRL+GNN agent, trained in Nsfnet, on 232 real-world topologies obtained from the Topology Zoo [43] dataset. Specifically, we take all the topologies that have up to 100 nodes. In Table 3.2 we can see the features extracted from the resulting topologies. The diameter feature corresponds to the maximum eccentricity (i.e., maximum distance from one node to another node). The ranges of the different topology features indicate that our topology dataset contains different topology distributions. For example, the minimum and maximum values for the variance of the node degree indicate that we can have topologies where all nodes have the same number of neighbors, or topologies where there are few nodes with a very high number of neighbors.

We executed 1,000 evaluation episodes and computed the average reward achieved by the DRL+GNN agents, the LB, and the theoretical fluid routing strategies for each topology. Then, we computed the relative performance (in %) of our agent and the LB policy with respect to the theoretical fluid model. Figure 3.8b shows the results where, for readability, we sort the topologies according to the difference of score between the DRL+GNN agent and the LB policy. In the left side of the figure we observe some topology samples where the scores of all three routing strategies coincide. This kind of behavior is normal in topologies where for each input traffic demand, there are not many paths to route the traffic demand (e.g., in ring or star topologies). As the number of paths increases, routing optimization becomes necessary to maximize the number of traffic demands allocated.

We also trained a DRL+GNN agent only in the Geant2 topology. The mean relative score (with respect to the theoretical fluid) of evaluating the model on all real-world topologies was +4.78%. These results indicate that our DRL+GNN architecture generalizes well to topologies never seen during training.

These experiments show the robustness of our architecture to operate in real-world topologies that largely differ from the scenarios seen during training. Even when trained in a single 14-node topology, the agent achieves good performance in topologies of up to 100 nodes.

Feature	Minimum	Maximum
Num. Nodes	6	92
Num. Edges	5	101
Avg. node degree	1.667	8
Var. node degree	0.001	41.415
Diameter	1	31

Table 3.2: Real-world topology features (minimum and maximum values).

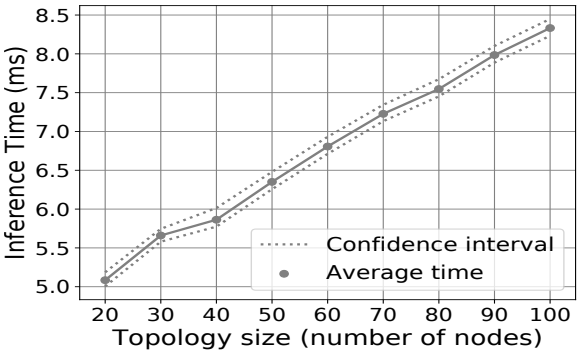


Figure 3.9: DRL+GNN average computation time (in milliseconds) over different topology sizes.

3.5.2 Computation Time

In this section we analyze the computation time of an already trained DRL+GNN agent when deployed in a realistic scenario. For this purpose, we used the synthetic topologies generated before in Section 3.5.1, and we executed 1,000 episodes for each one and we measured the computation time. This is the time the agent takes to select the best path to allocate all the incoming traffic requests. For this experiment we used *off-the-shelf* hardware without any specific hardware accelerator (64-bit Ubuntu 16.04 LTS with processor Intel Core i5-8400 with 2.80GHz $\times$ 6 cores and 8GB of RAM memory). Results should be understood only as a reference to analyze the scalability properties of our solution. Real implementations in a network device would be highly optimized.

Figure 3.9 shows the computation time for all episodes. The dots correspond to the average agent operation time over all the episodes and the confidence interval corresponds to the 5/95 percentiles. The execution time is in the order of few *ms* and grows linearly with the size of the topology. This is expected due to the way the message-passing in the GNN has been designed. The results indicate that, in terms of deployment, the proposed DRL+GNN agent has interesting features. It is capable of optimizing unseen networks achieving good performance, as optimization algorithms, but in one single step and in tens of milliseconds, as heuristics.

Typically, network optimization techniques are either based on optimization algorithms [39, 42] or traditional heuristics. While optimization algorithms can achieve good performance, this usually comes at the cost of high computation time, since they have exponential worst-case performance. On the other side, heuristics are fast but often result in limited performance. In terms of deployment, the proposed DRL+GNN agent has interesting features. It is capable of optimizing unseen networks achieving good performance, as optimization algorithms, but in one single step and in tens of milliseconds, as heuristics.

Topology Size	Mean Node Degree	Var. Node Degree	Node Betwee.	Edge Betwee.	DRL+GNN Perf. w.r.t. Fluid (%)
Nsfnet (training)	**3**	**0.2857**	**0.0952**	**0.1020**	**-**
20 Nodes	2.90	0.1050	0.1036	0.0988	4.305
30 Nodes	2.93	0.0956	0.0844	0.0764	-0.649
40 Nodes	2.95	0.1025	0.0704	0.0623	-3.945
50 Nodes	2.96	0.1104	0.0620	0.0538	-6.422
60 Nodes	2.97	0.1056	0.0559	0.0476	-8.103
70 Nodes	2.97	0.0920	0.0522	0.0437	-10.064
80 Nodes	2.98	0.0956	0.0474	0.0395	-11.380
90 Nodes	2.98	0.1062	0.0436	0.0361	-13.610

Table 3.3: Features for the Synthetic network topologies. The values correspond to the mean of all topologies from each topology size. As a reference, the first row corresponds to the Nsfnet topology used during training.

3.5.3 Discussion

In this chapter we proposed a data-driven solution to solve a routing problem in OTN. This means that our DRL agent learns from data that is obtained from past interactions with the environment. This method has the main limitation that when evaluated on out-of-distribution data, its performance is expected to drop. In our scenario, out-of-distribution is any data related to network topology, link features and traffic matrix that is radically different from the data seen during the training process.

The experimental results on synthetic and real-world topologies (Section 3.5.1 and Section 3.5.1 respectively) show that the DRL+GNN architecture has performance issues on some topologies. This performance drop is related to the diverging network characteristics from the topology used during training. The link features are normalized and the traffic demands always have the same bandwidth values, which excludes them as the source of the performance drop. However, the network topology changes, which has a direct impact on the DRL agent performance.

Table 3.3 shows different topology metrics for each topology size (in number of nodes). The edge betweenness is computed in the following way: for each edge compute the sum of the fraction of all-pairs of shortest paths that pass through the edge, and then make the mean of all edges. This applies in a similar way to the node betweenness. In addition, the DRL+GNN's performance with respect to the Theoretical Fluid model is also shown. The values correspond to the means from evaluating on all network topologies for each topology size (i.e., the means from the results in Figure 3.8a).

Topology Id	Avg Node Degree	Var Node Degree	Node Betwee.	Edge Betwee.	DRL+GNN Perf. w.r.t. Fluid (%)
Nsfnet (training)	**3**	**0.29**	**0.0952**	**0.1020**	-
0	2.42	9.59	0.0410	0.0484	-27.357
1	3.51	15.17	0.0447	0.0394	-23.944
2	2.00	41.41	0.0180	0.0298	-25.965
229	2.31	0.98	0.1294	0.1615	19.066
230	2.06	1.75	0.1140	0.1340	18.570
231	2.07	2.22	0.0994	0.1244	27.430

Table 3.4: Features for the real-world network topologies. The relative performance is the mean of 1,000 evaluation episodes. As a reference, the first row corresponds to the Nsfnet topology used during training.

Even though the synthetic topologies were generated in a way to have a similar node degree like Nsfnet, we can see that other metrics diverge as the topologies become larger. Specifically, the node and edge betweenness become smaller, which indicates that the pairs of shortest paths are more distributed. In other words, for small topologies the nodes and edges have proportionally more shortest paths crossing them than for larger ones. The network metrics clearly indicate that the more different the topologies are than Nsfnet, the worse is the DRL's performance.

Table 3.4 shows a similar table but for the real-world topologies. In this case, the performance results correspond to the means from the results in Figure 3.8b. Following a similar reasoning, we can see that the real-world topologies where the DRL+GNN architecture achieves the worst performance are radically different from Nsfnet (i.e., top left topologies from Figure 3.8b). In addition, we visualized the topologies with ids 0, 1 and 2 and observed that they correspond to topologies that have some nodes with a very high connectivity (see the variance of the node degree in Table 3.4). Similarly to the synthetic topologies, we again observe that the more different the topologies are than Nsfnet, the worse is the DRL+GNN's performance.

There are several things that could be done to improve the generalization capabilities for such topologies. A straightforward approach would be to incorporate topologies with different characteristics to the training set. In addition, the DRL+GNN architecture could be improved using fine-tuned traditional Deep Learning techniques (e.g., regularization, dropout). Finally, the work from [93] suggests that aggregating the information of the neighboring links using a combination of mean, min, max, and sum of the links' states improves generalization. We consider that improving the generalization is outside the scope of our work and we left it as future work.

3.6 Chapter Contributions

In this chapter we presented a DRL+GNN architecture for network optimization that is able to generalize to unseen network topologies. The use of GNNs to model the network topology allows the DRL agent to operate in different networks than those used for training. We believe that the lack of generalization was the main obstacle preventing the use and deployment of DRL in production networks. The proposed architecture represents a first step towards the development of a new generation of DRL-based products for real-time network optimization.

In order to show the generalization capabilities of our DRL+GNN solution, we selected a classic problem in the field of optical networks. This served as a baseline benchmark to validate the generalization performance of our architecture. The experimental results showed that the proposed DRL+GNN agent was able to generalize to the unseen Geant2 topology (see Section 3.4.3). However, SoA DRL-based solutions that use traditional neural network architectures were not able to generalize to other topologies. In addition, the DRL+GNN architecture is robust to operate in real-world topologies that largely differ from the scenarios seen during training (see Section 3.5.1). The source code, together with all the training and evaluation results are publicly available [80].

Chapter 4

Optimization in IP Networks

4.1 Introduction

In this chapter we improve the previous network optimizer architecture based on DRL and GNN and we evaluate it in a more challenging and realistic scenario. To do this, we investigated the related work and we moved to a relevant routing optimization scenario in IP networks. Then, we improved the DRL+GNN architecture and we added a second optimization stage to find better routing configurations at a small overhead. The experimental results indicate that our solution can achieve close-to-optimal performance in less than 30 seconds for a set of arbitrary real-world network topologies.

This chapter focuses on routing optimization in Wide Area Networks (WAN), a key infrastructure in today's society. During the last years, WANs have seen a considerable increase in network's traffic and network applications, imposing new requirements on existing network technologies (e.g., low latency and high throughput). To efficiently manage WAN infrastructures, operators take advantage of optimization techniques. Network optimization aims to efficiently manage the network resources by steering traffic to achieve a certain goal, for instance minimizing the utilization of the most congested link. In our work, the optimization problem is defined by the network infrastructure, the traffic matrix, the routing and the link capacity (Section 4.2.1). Similarly to the OTN scenario, the NDT of the WAN is implemented using Python as we work with per-link utilizations. These can be easily obtained by adding the traffic crossing on each link from the network topology.

WANs have recently been *softwarized*, this is referred to as SD-WAN [94]. SD-WANs offer programmability and the SDN controller has a full view over the network resources, enabling a new breed of centralized optimization algorithms. A notable example is DEFO [42], which uses a centralized constraint programming algorithm to produce solutions in a few minutes. The centralization and softwarization of the network has allowed it to achieve unprecedented performance [95].

In our work we explore the feasibility of designing a DRL-based method for solving complex optimization problems in WANs. We propose *Enero*[1], a real-time high performance optimization engine that implements the DRL+GNN architecture (Section 4.3). In addition to DRL, we use a Local Search (LS) algorithm to improve DRL's solution at a small overhead(Section 4.3.1). Intuitively, LS explores the solution space by applying small changes to the DRL's solution. In contrast to other existing solutions, our method does not require the network operators to design hand-crafted heuristics nor to use expert knowledge.

Several works analyzed WAN's traffic behavior to study and model it [45–47]. These studies found that the significant changes in traffic patterns happen frequently, on the scale of several minutes. Thus, to be able to solve an optimization problem before the traffic changes significantly, we considered real-time to be in a sub-minute time scale.

One of the problems of using DRL in real-world scenarios is that it does not offer performance bounds. This means that once a DRL agent is trained, there is no way to give a minimum certainty over the DRL agent's performance. This performance bound would let network operators know when the DRL agent's performance is poor before deployment and avoid compromising the real network's behavior. Consequently, network operators typically do not feel confident to deploy such technology in a real-world network. In our work, we designed a method to offer a minimum performance certainty or bound in the DRL agent's performance (Section 4.3.2).

Another important characteristic is that the proposed solution is able to adapt to changes. WANs suffer from changes constantly, physical links can be broken due to external factors and network users have different pattern behaviors that cause difficult-to-predict spikes in network's resources utilization. When such an event occurs, SoA optimization solutions based on heuristics or classical optimization techniques need to start the training and optimization processes from scratch.

In our work, Enero is designed with a DRL agent that incorporates a GNN [65]. By using a GNN in the DRL agent, we enable Enero to operate efficiently over different network scenarios when the traffic matrix or the network topology changes during time.

In summary, our work in this chapter makes the following contributions:

- We propose Enero, a two-staged method that implements a more advanced DRL+GNN architecture and a LS algorithm to reach high quality optimization solutions in real-time (Section 4.3.1).
- We propose a method to offer a performance certainty or lower bound in the DRL agent's operation (Section 4.3.2).
- We design a DRL agent that is able to operate efficiently while link failures occur and is able to adapt to dynamic traffic matrices (Section 4.3.3).

[1] EfficieNt rEal-time Routing Optimization

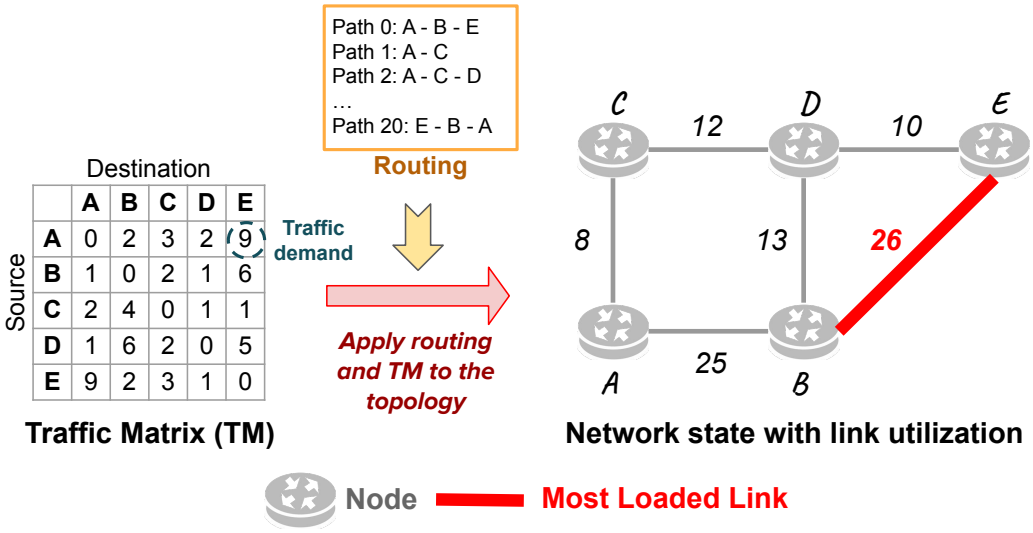


Figure 4.1: The routing configuration is applied to the traffic matrix and is combined with the network topology, resulting in a network state with link utilization values. Our goal is to minimize the utilization of the most loaded link.

4.2 Background

4.2.1 Problem Statement

The problem we want to solve corresponds to the classic Traffic Engineering (TE) problem of minimizing the maximum link utilization [41, 42, 96, 97]. This is because we are interested in avoiding sending packets over congested links. A congested link is where the amount of traffic crossing the link is larger than the link capacity. When this happens, the excess packets are dropped, causing packet losses. Thus, we want to minimize the most congested link and to efficiently use the network's resources.

The TE problem is defined by a directed graph, a Traffic Matrix (TM) and an initial routing configuration. We abstract the real-world network topology as a directed graph, where the physical routers are represented by nodes with no features associated. Between two nodes there are always two links which correspond to the upstream and downstream links. In reality there can be multiple links between two nodes. However, we abstract from such technicality and we aggregate all the capacities into a single link for each direction. The traffic matrix indicates the volume of traffic that is being sent through the network. Specifically, the TM has size NxN where N is the number of nodes. Each pair of nodes *(s, d) with* $s \in N$ *and* $d \in N$ corresponds to a *traffic demand* which is an aggregate of flows. In our optimization scenario, we do not take into account the traffic demands with $s==d$ (i.e., the nodes do not send traffic to themselves). Figure 4.1 illustrates how the routing and the TM are combined to obtain a network state with the link utilization values.

Initially, each traffic demand is allocated using the OSPF protocol with unitary link weights. These weights are initially assigned by the network operator using different meth-

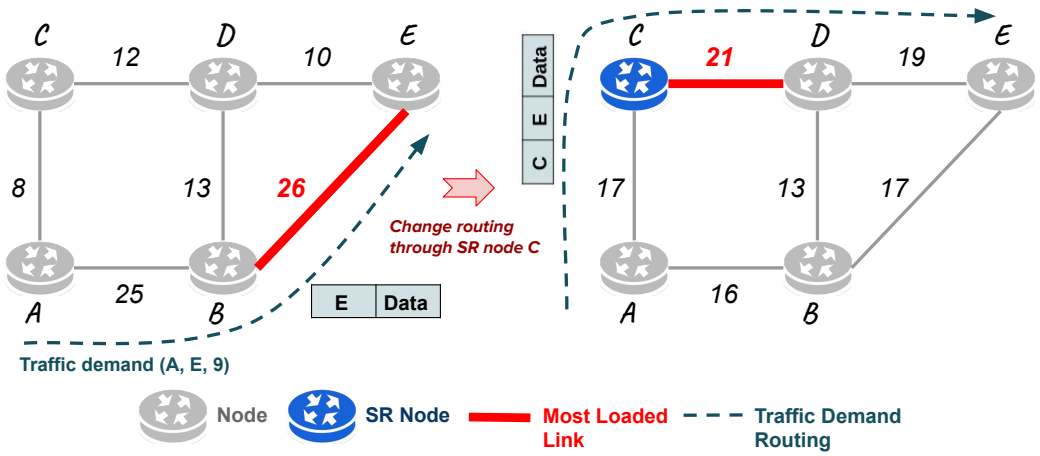


Figure 4.2: Single step optimization process that illustrates how changing the routing of the traffic demand (A, E) minimizes the maximum link utilization. Specifically, traffic demand (A, E) is assigned with the intermediate SR node C to program a detour and avoid link B-E.

ods (e.g., unitary weight or inverse of the link capacity). Then, the goal is to change the routing policy such that the maximum link utilization is minimized. Ideally, the final solution should decrease the link utilization in a way that the amount of traffic volume crossing the most loaded link is below the link's capacity.

We leverage Segment Routing (SR) [98] to enable fast and efficient centralized network management. SR is a protocol that includes routing related-information in the IP packet headers. This means that each packet will have an SR path to reach a destination node. Then, SR Ingress routers encapsulate incoming packets to create a tunnel that traverses an SR path before reaching their respective destination. This SR path is composed of different segments, and in each of them, the endpoint node removes the outermost encapsulation label. This process is repeated until the packet reaches the SR Egress node. The packets within a segment are routed using the traditional OSPF routing protocol. In TE terms, SR can program detours in forwarding paths so that network packets avoid crossing congested links. Previous work showed that SR using 2-segment paths offers enough flexibility to achieve high network performance [97]. In our work, we adapt a similar approach and we consider only one intermediate node between SR Ingress and Egress nodes.

Figure 4.2 shows an illustrative example of the TE problem we want to solve. In the figure, the overlay routing for a single traffic demand is changed. Specifically, the traffic demand that goes from node A to node E has a bandwidth of 9 and it initially uses OSPF to reach the destination. This corresponds to the left-hand side network state from the same figure. Then, a good action to minimize the maximum link utilization would be to re-route the traffic demand through the intermediate node C. This means that the SR path would be A - C - E, where C is the intermediate node. This process is repeated for all the

traffic demands (i.e., all pairs of source-destination nodes), where their routing policies are changed such that the maximum link utilization is minimized.

4.2.2 Shortcomings of Existing Solutions

The TE problem can be formulated as an Integer Linear Programming (ILP) problem and can be solved using SoA optimizer engines such as Gurobi [37] or CPLEX [38]. In our TE problem, the decision variables correspond to the traffic demands and the link capacities constrain the optimisation problem and define the solution space. There is at most one traffic demand for each pair of nodes. When the problem size grows (i.e., the number of nodes and links grows), the number of decision variables increases and the solution space becomes larger and more complex. In this context, TE in WANs results in a large combinatorial space where the number of possible routing configurations for each traffic demand explodes. Consequently, ILP solvers would take several weeks to find the exact solutions in WANs as they have in the order of hundreds of links and nodes [42, 43].

An alternative to ILP is the use of Constraint Programming (CP) [44]. This method defines the combinatorial problem to solve with a set of decision variables (e.g., traffic demands, OSPF weights), a set of domains (i.e., potential values of the decision variables) and a set of constraints on the feasible solutions (e.g., maximum link utilization must be below a threshold). To define the constraints that limit the solution space and makes it tractable is non-trivial in WANs due to their size and complexity. In addition, the user indicates some time limit and the solver will return the best solution found within the specified time (e.g., DEFO [42]). Therefore, when solving a TE problem using CP, network operators should estimate the solver's execution time needed to obtain a solution with the desired performance. However, WANs experience external events frequently (e.g., link failures, increase in traffic demand), altering the normal network behavior [45–47]. This method has the limitation of finding sub-optimal solutions if the specified time is not long enough.

Finally, network operators can use heuristics or expert knowledge to design an algorithm to solve TE problems. In addition, they can leverage heuristics to reduce the problem dimensionality by pruning the solution space, and then use a traditional method to solve the smaller problem (e.g., CP, ILP). In the last years, WANs' size and traffic have been growing by almost doubling every year [2–4], raising the complexity of efficient network operation. As a result, the design of high performance heuristics for TE became more challenging for humans, and with a higher cost for network operators. In addition, human experts typically use trial-and-error processes that can take several months, which does not scale with recent trends in WANs.

4.2.3 Deep Reinforcement Learning for Traffic Engineering

TE is a multi-step decision making process where the decisions have long term effects (i.e., the assignment of a routing policy to a traffic demand is known to be good or bad once iterated over all traffic demands). Similarly, DRL is a technology capable of modeling future rewards. This means that DRL can optimize the routing configuration taking into account the future. That is to say, DRL can learn a long-term routing policy by taking into account the future expected rewards. For example, to change a routing policy of a traffic demand might not lead to an immediate minimization of the maximum link utilization but to a delayed one that the DRL agent will observe later in the future. This is contrary to heuristics where they can not establish a relationship between local decisions (e.g., change a routing configuration for a traffic demand) and long-term strategies to solve an optimization problem (e.g., minimize the most loaded link), leading heuristics to achieve sub-optimal performance. The long-term planning capabilities make DRL a key technology for solving the TE problem.

The TE problem can be seen as a combinatorial problem where traffic demands are assigned to routing policies such that the utilization of the most loaded link is minimized. The difficulty of combinatorial problems can make the DRL reach sub-optimal solutions. The reason behind this is that when the DRL agent makes a bad decision, it has no way to undo it and explore other actions. To solve this issue, we incorporated a low computational overhead optimization step that is executed after the DRL's agent optimization process.

4.3 Proposed Solution

Enero is a two-stage method for real-time routing optimization that combines DRL and LS. In the first stage, Enero leverages DRL to find a good initial solution to the TE problem by taking into account future traffic demands. Recall that we consider a traffic demand as a source-destination node pair with a bandwidth that represents an aggregate of flows between the node pair. In the second stage, Enero tries to improve DRL's solution using a LS technique. In our work, the LS step implements the hill climbing heuristic that behaves in a greedy way by making incremental changes to the DRL solution.

Intuitively, LS explores the solution space by applying small changes to the initial configuration or solution. The motivation behind the combination of DRL with LS is to leverage DRL's long-term planning capabilities and to improve DRL's solution using LS. Combining DRL with traditional optimization techniques has shown to achieve high performance in complex scenarios [99, 100]. We believe that DRL and LS complement each other, increasing the performance of the returned solutions.

The number of traffic demands grows quadratically with the number of nodes in a network. For instance, in a topology with 30 nodes there are $30 * 29 = 870$ traffic demands whose routing needs to be reconfigured to solve the TE problem. Ideally, we would like

to take into account all traffic demands in our TE optimization problem to ensure that our solver can find the best routing configuration. However, the solution space becomes intractable for large TE problem instances and computationally expensive even when using heuristics. Inspired by [100], we decided to take a subset of these traffic demands. These are called *critical demands* and they are selected from the set of traffic demands crossing the 5 most loaded links. We initially performed some experiments where we optimized selecting different percentages of the traffic demands (i.e., 10, 15, 20 and 50). The results showed that taking 15% of the critical demands offered the best trade-off between computation time and performance.

4.3.1 Two-stage Optimization

The complexity of the combinatorial problem can make the DRL agent achieve sub-optimal routing configurations. This is because, on the contrary to some existing solutions that use backtracking (e.g., DEFO [42]), the DRL agent has only one shot to converge to the optimal solution (i.e., a single iteration over all traffic matrices). To solve this issue, we improve DRL's solution using a LS technique without adding a large computational overhead.

The LS step implements a hill climbing heuristic. This method makes small incremental changes to the DRL's solution, trying to find new TE solutions that are fundamentally close to DRL's resulting configuration. Specifically, LS iterates over all traffic demands and all possible SR paths, trying to find which is the best configuration that minimizes the maximum link utilization. Similarly to the DRL case, LS iterates only over 15% of the critical traffic demands. We decided to adapt LS in the second stage for being an anytime optimization technique. This means that the LS search process can be stopped at any moment and the result returned will always be a valid one.

4.3.2 Performance Lower Bound

Even though DRL is a key technology to learn long-term strategies, it can still make mistakes. DRL is a data-driven method and when evaluated in out-of-distribution data (i.e., data totally different than the one used in the learning process), it is to be expected that the performance will degrade. In our TE problem this can happen due to different bandwidth scale values in the traffic matrices, due to different extreme topologies that can radically change the action space or because of the high complexity of exploring the solution space.

To solve this problem and to enable the deployment of the DRL technology in real-world scenarios, we had to give some minimum performance certainty for the DRL agent. With this lower bound, the network operator can know for certain the DRL agent's performance before deploying it on the real network. To do this, the DRL agent starts the optimization process from a predefined routing policy. In our work we consider OSPF as

the starting routing policy. Then, it starts the optimization process and changes the routing configuration of the critical traffic demands. If the DRL agent is not capable of minimizing the maximum link utilization, it returns the initial routing configuration to the LS stage. Enero is designed to allow the starting routing policy to be initialized using any routing policy (e.g., expert-knowledge, heuristic-based routing policy).

4.3.3 Deep Reinforcement Learning Agent

The DRL setup can be described by defining the environment state, the action representation and the reward. The *environment state* includes the NDT of the WAN of our optimization problem. The NDT is defined by a network topology with the links' features (i.e., link capacity and utilization). Similarly to the OTN scenario, the NDT is implemented in Python because we work with per-link utilizations. This can be easily simulated by adding the traffic crossing on each link. Notice that some optimization problems requires the NDT to be implemented with sophisticated ML models that mimic complex behaviours such as delay or jitter.

When the DRL agent performs an action (i.e., applies a new routing policy to a given traffic demand), the link's utilization is updated. The *action* is represented directly on the network topology by marking the links that are part of the action. In other words, the links of the path going from a source node to a SR intermediate node and from this to the destination node are marked with a flag. All the nodes from the topology can be SR intermediate nodes. This process is repeated for each possible action of the current traffic demand whose routing needs to be changed. The DRL's GNN is then in charge of

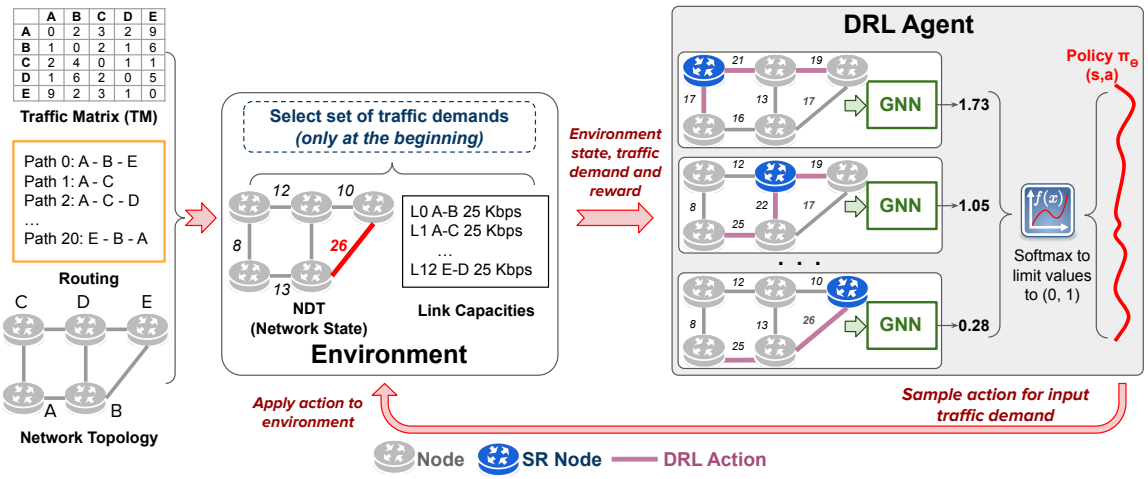


Figure 4.3: The *critical demands* are computed in the GYM Environment at the beginning of an episode. Then, the DRL agent iterates over them and for each demand, the agent explores the action space by marking the links for each SR path. Afterwards, a GNN takes the graphs with the actions marked and outputs a probability distribution. The action to perform is sampled from the distribution and applied to the environment.

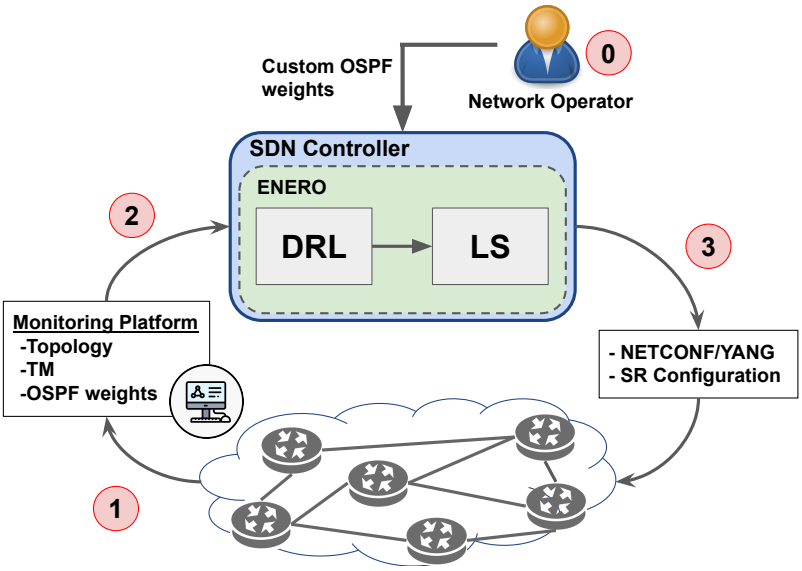


Figure 4.4: Enero's workflow.

processing these graphs (i.e., one graph per action where the action is marked directly on the links) and will output a probability distribution over the actions. Finally, the reward is the difference between the maximum link utilization between two steps. This difference is relative to the link capacities. Figure 4.3 shows an overview of the DRL setup and the operation process.

4.3.4 Workflow

Enero is an optimization engine that is placed in the SDN controller. It takes as input the network topology, the TM and the initial routing configuration. When the SDN controller detects a change in the network (e.g., traffic matrix changed, link failure), it executes Enero to start the optimization process for the new scenario. This process finishes in under a minute, enabling real-time operation.

Figure 4.4 shows *Enero's* step-by-step workflow. At the beginning (Step 0), the network operator defines the initial OSPF weights. These weights are used to initially route the traffic demands and to route the demands within SR segments [97]. Their values can be assigned either by the network operator using heuristics and expert knowledge or by using some well-established OSPF weights initialization (e.g., unitary weight values, inverse of the link capacity).

Once the initial routing policy is defined, a monitoring platform is in charge of retrieving the relevant information for the TE optimization problem (Step 1). This information consists of the network topology, the TM and the OSPF weights. Then, Enero takes this information (Step 2) and starts the optimization process. When the process finishes, the routing configuration (i.e., per-demand SR intermediate node assignment) is pushed down to the data plane (Step 3). This means that each traffic demand is going to be assigned

a SR intermediate node. When there is some change in the data plane (e.g., the topology or the TM changed), the monitoring platform will detect these changes and will launch Enero again to optimize the new scenario. There are many efforts put on the design of fast and efficient monitoring platforms and we consider it to be outside the scope of this work [101–103].

4.3.5 Training Algorithm

The DRL agent training is an iterative process that takes as input a network topology, a set of traffic matrices, the links' features and the initial OSPF weights defined by the network operator. Then, the DRL agent will learn how to optimize over the given routing configuration and for different TMs. To do this, the DRL agent iterates over the traffic demands following a decreasing bandwidth order, changing the routing policy for each demand. This means that for each traffic demand, the DRL agent will assign the best SR intermediate node before reaching the destination node. This process can be seen as changing the direct path from the source node to the destination node by creating a detour. This is a trial-and-error process where at the beginning the agent will explore different routing configurations, and as the training advances, the agent will tend to exploit more of the action space.

Algorithm 4 shows the pseudo-code of the actor-critic training process. For the sake of simplicity, the pseudo-code describes the training process using a single network topology. The same process can be applied to multiple topologies by repeating the lines 3 to 9 for each topology. The training process starts in line 2 and finishes when the number of training episodes E has been reached. At the beginning of the training episode, the DRL environment

Algorithm 4 DRL Agent Training Process

1: **Input:** Network topology (G), link capacities, TMs, Initial OSPF Weights (OSPFw)
2: **for** i in $0, ..., E$ **do**
3: $env, d \leftarrow init\_env(G, TM, OSPFw)$
4: **while** not Done **do**
5: $act\_dist \leftarrow pred\_act\_distrib(env, d)$
6: $c\_val \leftarrow pred\_critic\_value(env)$
7: $a \leftarrow choose\_action(act\_dist)$
8: $d, Done, r \leftarrow step(a, d, env)$
9: $store\_results(act\_dist, c\_val, a, d, Done, r)$
10: $c\_val \leftarrow pred\_critic\_value(env)$
11: $ret, adv \leftarrow compute\_GAE(c\_val', r')$
12: $a\_loss \leftarrow compute\_actor\_loss(adv)$
13: $c\_loss \leftarrow compute\_critic\_loss(ret)$
14: $total\_loss \leftarrow a\_loss + c\_loss - entropy$
15: $grads \leftarrow compute\_gradients(total\_loss)$
16: $clip\_gradients(grads)$
17: $apply\_gradients(grads)$

is initialized (line 3). This means that the NDT is built and the per-link utilization is updated according to the initial OSPF routing policy.

The loop from line 4 indicates the iteration of the DRL agent over the critical traffic demands. In each loop iteration, the DRL agent tries to change the routing policy of a single traffic demand (i.e., assign an SR intermediate node). In line 5 the DRL agent uses the GNN to output a probability distribution over the action space. Then, the critic network predicts the value of the current state.

The DRL agent uses a random sampling of the action distribution to pick the action to perform (line 7). During evaluation, the sampling is changed by taking the action with higher probability. Then, the selected action is sent to the environment to be applied over the current network state and to update the link's utilization. In line 9, the agent stores all the intermediate results that will later be used to compute the losses.

The next step is to compute the Generalized Advantage Estimates (GAE), which is a method to reduce variance in policy gradient algorithms [64]. Then, the actor and the critic losses are computed [61]. These are then combined in a sum and subtracted with the entropy term, used to guide the exploration during training [104]. Finally, the gradients are computed and clipped to avoid the policy to change too much for a given training step, and they are applied to the actor and critic networks.

4.4 Experimental Evaluation

In this section, we first describe the implementation details and the methodology used to obtain the datasets and to train the DRL agent. Then, we made an experimental study to see the performance gap between DRL, LS and Enero. Finally, we perform a series of experiments on different real-world network scenarios. Specifically, we want to answer the following questions:

- What is the performance gap between DRL, LS and Enero for solving TE problems? (Section 4.4.3)
- How does Enero perform when the traffic matrix changes during time? (Section 4.4.4)
- What is Enero's performance when the topology changes as a result of link failures? (Section 4.4.5)
- What is Enero's performance and execution cost compared with SoA TE solutions? (Section 4.4.6)

All the experiments were executed on off-the-shelf hardware without any specific hardware accelerator or high performance software optimization engine. Specifically, we used a machine with Ubuntu 20.04.1 LTS with processor AMD Ryzen 9 3950X 16-Core Processor.

Hyperparameter	Value
GNN Hidden State	20
Message Passing Steps	5
Evaluation Episodes per Topology	20
Training Epochs	8
% critical demands	15%
Mini-batch size	55
Learning Rate	0.0002
Decay Rate (Decay Steps)	0.96 (60)
Entropy Beta (After 60 Episodes)	0.01 (0.001)
GAE Gamma, Lambda	0.99, 0.95
Gradient Clipping Value	0.5
Actor L2 Regularization	0.0001
Readout Units	20
Activation Function	Selu

Table 4.1: Enero hyperparameter configuration.

4.4.1 Implementation

Enero's DRL stage (training and evaluation) was implemented using Tensorflow [85] and the DRL environment was implemented using the OpenAI Gym framework [86] and Python. Recall that the NDT of the network infrustructure is implement using Python and is contained within the DRL environment. The DRL agent is trained using the PPO algorithm [61]. The LS stage is implemented totally in Python except for some operations where it uses the Numpy library [105]. The LS execution cost could be improved by using more efficient libraries (e.g., Cython [106]), which were left as future work. For some graph-related operations the NetworkX library [92] is used. Table 4.1 shows the hyperparameters used during Enero's DRL agent training stage. Enero's code is publicly available [107].

4.4.2 Methodology

Traffic Matrices

The traffic matrices were generated using a uniform distribution. This means that the bandwidth values from the traffic demands were uniformly distributed from 0.5 to 1. Then, we scaled this value to obtain the TM's bandwidths in Kbps and to have the same unit for both bandwidth and link capacities. Each network topology had a total of 150 TMs.

Network Topologies

We obtained the network topologies from the TopologyZoo dataset [43], which contains real-world topologies from network operators. Specifically, we took all topologies up to 100 links and 30 nodes, resulting in a total of 74 topologies. From these topologies, only 3 of them were used in the DRL agent's training process. In our TE problem we only consider the link capacities, which means that nodes do not have any features associated.

DRL Agent Training

In all the experiments we are always evaluating the same DRL agent. This means that we have only trained *a single DRL agent* and incorporated it into Enero. To train the DRL agent, we arbitrarily picked 3 network topologies (i.e., BtAsiaPac, Garr199905 and Goodnet topologies) from the 74 topologies extracted from the Topology Zoo dataset. We split the original 150 TMs from each topology into 100 TMs for training and 50 TMs for evaluation. During training, the DRL's agent performance evolution is evaluated on the BtAsiaPac, Garr199905 and Goodnet topologies after every training step. Specifically, the agent is evaluated on 20 TMs uniformly sampled from the evaluation split for each topology.

Comparison Baselines

We compare Enero with three baselines which together represent widely used heuristics and close-to-optimal solutions. The first baseline is the Shortest Available Path (SAP) heuristic. SAP starts with the empty network and iterates over all traffic demands in decreasing order of bandwidth. This is done to allocate the bigger and critical traffic demands first. Then, each traffic demand is routed using the path with the highest available bandwidth. The second baseline corresponds to a LS algorithm. Specifically, we implemented the hill climbing search to improve an initial routing configuration in a greedy fashion. Similarly to Enero, this method starts in the same routing configuration using the OSPF protocol and tries to minimize the maximum link utilization. This is an iterative process where in each step applies the routing policy of the traffic demand that minimizes the maximum link utilization. This process finishes when the maximum link utilization does not improve anymore.

To compute the optimal solution for our TE problem it would require weeks of computation using ILP. As it is not feasible to do that, we chose DEFO [42] as our close-to-optimal baseline. In particular, we took the implementation from [108] and adapted it to have at most one intermediate SR node per traffic demand. DEFO is a CP-based solution and if left enough time executing it provides a close-to-optimal solution. This is the reason why we left DEFO executing for 180 seconds in all of our experiments. Following the recommendations from the experiments in the original paper [42] on very large topologies (i.e., a few hundreds of links and more than 6.000 traffic demands to optimize), we expect that 180 seconds is enough to find close-to-optimal solutions in our topologies (i.e., we have topologies of up to 100 nodes and 900 traffic demands).

DEFO uses Equal-Cost Multi-Path routing (ECMP) to route the traffic demands. This enables DEFO to divide the traffic demands among multiple paths, achieving a better traffic distribution and a lower link utilization. In our problem setup the traffic demands are routed using solely a single path, creating a natural gap between DEFO and Enero's performance. We left the task of enabling Enero to optimize using ECMP for future work.

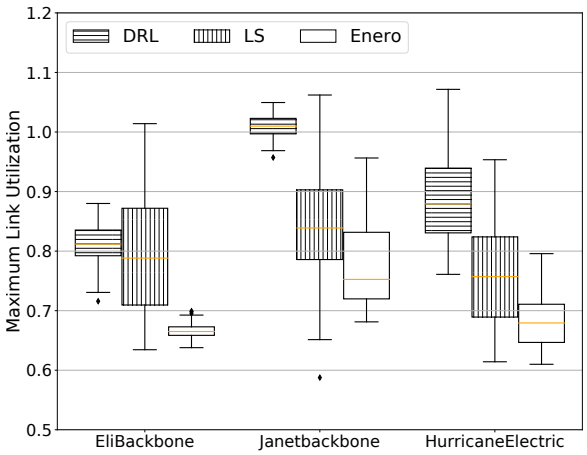


Figure 4.5: Performance of LS, DRL and Enero for the EliBackbone, Janetbackbone and HurricaneElectric topologies.

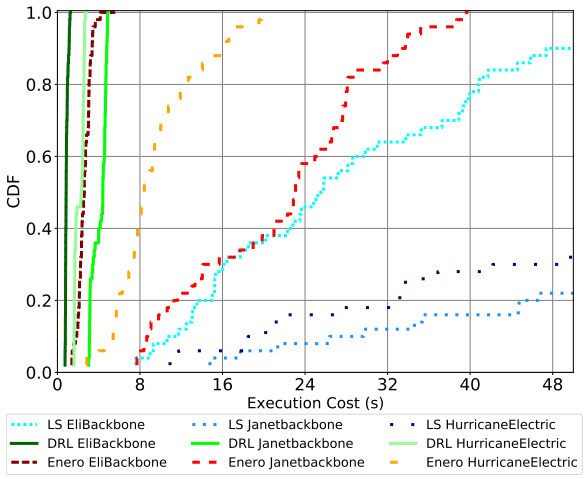
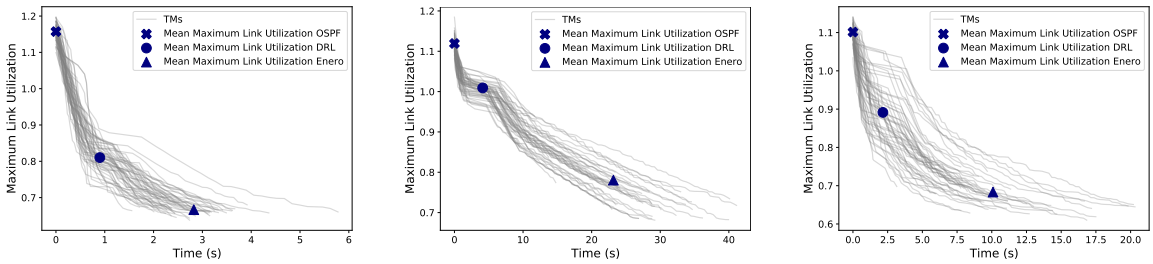


Figure 4.6: Execution cost of LS, DRL and Enero for the EliBackbone, Janetbackbone and HurricaneElectric topologies. Best viewed in color.

4.4.3 DRL and LS Hybrid Method

In this section we want to demonstrate the capabilities of combining DRL with LS. To do this we studied the performance and execution cost of DRL and LS individually and compared them with Enero. In the experiments, we evaluated the DRL agent, the LS algorithm and Enero on three network topologies using 50 TMs per topology. Figures 4.5 and 4.6 show the resulting performance and the CDF of the execution cost respectively. Notice that the topologies from these figures were not seen by the DRL agent during the training process.

The experimental results indicate that DRL has a reasonably good performance in all three topologies. This is because it can minimize the maximum link utilization from ≈1.1 to below 1 for EliBackbone and HurricaneElectric topologies and to ≈1 for the Janetbackbone topology. LS can minimize the maximum link utilization in all three topologies, obtaining



(a) EliBackbone (b) Janetbackbone (c) HurricaneElectric

Figure 4.7: Dynamic traffic matrix scenario. Enero evaluation on different real-world network topologies. For each topology, we evaluated over 50 different TMs. Notice that the topologies from this figure were not seen by the DRL agent during the training process.

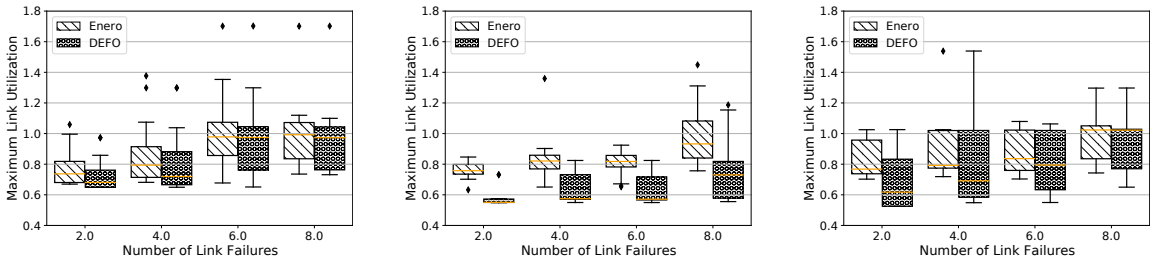
better performance than DRL. However, the CDF from Figure 4.6 indicates that the DRL is extremely fast while LS takes a considerable amount of time (up to minutes).

To demonstrate the capabilities of combining DRL with LS we also plot in Figures 4.5 and 4.6 Enero's performance and execution cost respectively. The results indicate that Enero reaches better TE solutions than DRL and LS in all three topologies while the execution time is below 40 seconds for the Janetbackbone topology. Notice that the Janetbackbone topology is a large topology with 812 traffic demands whose routing policy needs to be optimized, which explains the larger execution times.

4.4.4 Dynamic Traffic Matrix

In this scenario we evaluated Enero's performance when the traffic matrix changes during time. In our experiments we took the extreme case where every 60 seconds the entire TM changes. The reason behind this is to simulate the worst-case scenario where Enero must re-compute the solution to the TE problem from scratch. We repeated this process until the TM has changed 50 times.

Figure 4.7 shows the evaluation results on three network topologies with 50 TMs per topology. Each line indicates the progress of the maximum link utilization while Enero is solving the TE problem for a given TM. In reality, the lines should be concatenated one after another but for visualization purposes we aggregated all the events where the TM changed into a single figure per-topology. From the same Figures we can observe Enero's two-stage optimization process. When the monitoring platform detects a change in the TM (see Section 4.3.4), Enero uses the pre-defined OSPF routing policy and then starts the optimization process. We can appreciate that in all topologies the DRL agent quickly finds a good TE solution and then LS improves it. Notice that the topologies are different from those used during the DRL agent training process. This showcases Enero's capabilities to perform TE on different network topologies (than those seen during training) and with dynamic changes in the TM.



(a) EliBackbone (b) Janetbackbone (c) HurricaneElectric

Figure 4.8: For each number of link failures there are 20 different topologies and we evaluated using 50 TMs for each topology.

4.4.5 Link Failures

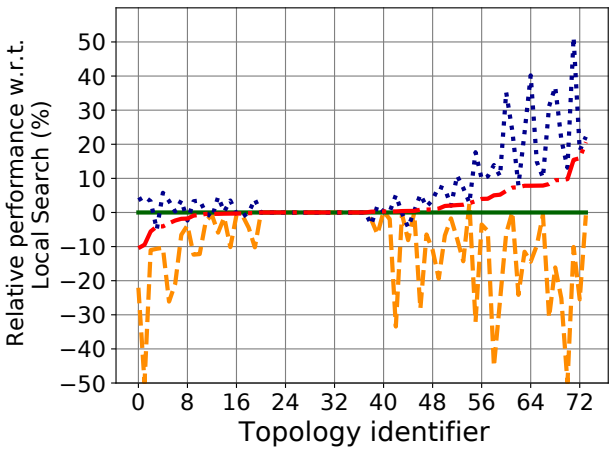
In this experiment we evaluated Enero's capabilities to react to changes in the network topology resulting from link failures. We simulated link failures by randomly removing links from the topology in each of the evaluation topologies. We made sure that there are no two topologies that are the same after removing some links. For each logical link in the topology, there are the upstream and downstream links. To ensure network connectivity, when we drop a link we drop both upstream and downstream links. We simulated up to 8 link failures in total where for each failure there are 20 different topologies and for each topology there are 50 TMs.

To make the experiments more challenging, we used the original TMs from the topologies. In other words, the bandwidths from the TMs remained the same while link failures were happening. This means that while the traffic demands did not change, link failures forced the network to have less and less resources to accommodate the original TMs.

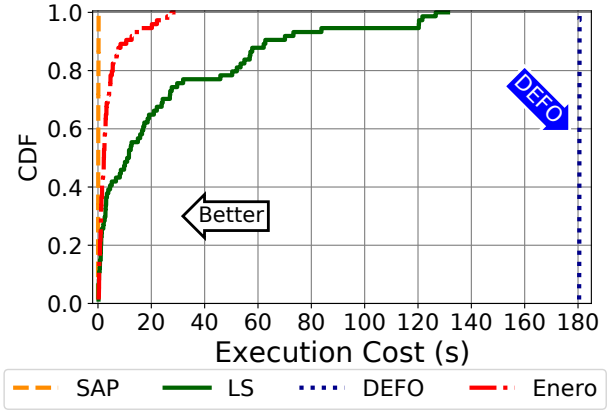
Figure 4.8 shows Enero's results after optimization for each link failure together with the results from DEFO and SAP baselines. Because the TMs did not change and the topology had less resources to accommodate the bandwidths, the maximum link utilization should be increasing when links from the topology fail. The results indicate that Enero's performance has a similar behavior to DEFO regardless of the number of link failures. Recall that DEFO is our close-to-optimal baseline which has been executed during 180 seconds and uses ECMP to split the traffic demands among multiple paths.

4.4.6 Operation Performance and Cost

In this experiment we wanted to evaluate Enero's performance while operating on a set of real-world topologies. To do this, we took all topologies from the TopologyZoo dataset that had up to 100 links and 30 nodes. This made a total of 74 topologies, from which only 3 of them were used in the DRL agent's training process. Figure 4.9 shows the evaluation results over all 74 topologies. Specifically, in Figure 4.9a we plot the relative performance with respect to the LS baseline. The topologies from 20 to 37 are ring, star or line topologies where there is no room for optimization. This explains why all the baselines have exactly



(a)



(b)

Figure 4.9: Relative performance (a) and CDF of the execution cost (b) on the TopologyZoo dataset. In sub-figure (a), the topologies from 20 to 37 are ring or star topologies where there is no room for optimization.

the same performance. Figure 4.9b shows the execution cost of all the baselines. As a reminder, DEFO was set to execute for 180 seconds to ensure a close-to-optimal solution. The results indicate that Enero is capable of obtaining better performance than the SAP and LS baselines and in most of the topologies has a similar performance to DEFO. In addition, Enero's execution cost is small, with only 5 topologies with an operation cost of more than 20s.

Enero is a data-driven solution that can use synthetic or real-world data to train a DRL agent to solve TE problems. This means that if we deploy our agent over topologies or TMs that are very different from those from the dataset used in the training process, we can expect our agent's performance to drop. This explains Enero's poor performance for the top left topologies from Figure 4.9a. Specifically, the traffic demand values are all limited by the uniform distribution between 0.5 and 1, meaning that the TMs can be discarded

Topology/Id	Node Degree	Edge Betweenness
BtAsiaPac	(2, 24, 6.2)	(0.010, 0.067, 0.04)
Goodnet	(2, 18, 7.3)	(0.014, 0.059, 0.03)
Garr199905	(2, 18, 4.35)	(0.0435, 0.083, 0.05)
0	(4, 8, 4.3)	(0.043, 0.167, 0.11)
1	(2, 14, 5.23)	(0.026, 0.117, 0.07)
2	(2, 8, 4.2)	(0.044, 0.164, 0.10)
3	(2, 6, 4.0)	(0.067, 0.162, 0.12)

Table 4.2: TopologyZoo metrics. For each topology and each metric the tuple values correspond to the *(min, max, mean)* values respectively. The top 3 topologies are those used during DRL's agent training process.

as the source of performance instability. Thus, we focused our attention on the network topologies and we wanted to study what is different (in connectivity terms) in the top left topologies in Figure 4.9a.

We identified two metrics that showcase the differences between the topologies used during training and those where Enero's performance is worse. The first one is the node degree, which indicates the number of adjacent links to a node. The second metric is the edge betweenness, which computes the portion of all pairs of shortest paths that pass through each link l of a graph [109]. The following equation describes the edge betweenness metric:

$$c(l) = \sum_{s,d \in N} \frac{\sigma(s, d|l)}{\sigma(s, d)} \tag{4.4.1}$$

where N is the set of all nodes, $\sigma(s, d)$ is the total number of shortest paths and $\sigma(s, d|l)$ is the number of shortest paths that pass through link l.

Table 4.2 shows the minimum, maximum and mean node degree and edge betweenness for each topology used during training and for the 4 topologies where Enero had worse performance. These metrics indicate that the topologies seen during training and the ones where our method performs worse are totally different. For example, the minimum and the average edge betweenness is much higher in the topologies 0, 1, 2 and 3. This indicates that the shortest paths are not well distributed and they cross the same links, making them become critical links for the TE problem. In addition, the topologies used in the training process have a higher average and a wider range of the node degree. This indicates that the nodes are more interconnected between them than in the topologies 0, 1, 2 and 3.

There are several ways to solve the out-of-distribution problem. For example, we could work with specific Deep Learning techniques such as regularization or dropout. However, the most effective way would be to add more data to the training process. This is translated

to our problem by adding more topologies to the DRL's training that are different between them.

The experimental results showed that the hybrid method of combining DRL with LS enables efficient real-time routing optimization. However, there is still room to push even further the combination of DRL with traditional optimization methods. The straightforward approach would be to improve the LS implementation using high performance software (e.g., Cython [106]). In addition, in our work we used a greedy approach in Enero's second stage but it could be substituted by more advanced search algorithms (e.g., CP, Genetic Algorithms). For example, the DRL's solution could be converted to constraints and then some CP solver (e.g., Gurobi [37]) could find a better solution. This would ensure that the solution of the CP phase should be better than the one from the DRL agent and it would enable the second optimization stage to explore better solutions.

4.5 Chapter Contributions

In this chapter, we extended the DRL+GNN architecture from the OTN scenario and we presented Enero, a method that combines the DRL+GNN architecture with LS to solve optimization problems in real-time. We evaluated Enero in a more realistic and complex optimization scenario of IP networks. The experimental results showed that Enero is able to operate efficiently in real-world scenarios (e.g., with dynamic traffic matrix, link failures). In addition, the results indicated that Enero can achieve close-to-optimal performance in less than 30 seconds for a set of arbitrary real-world network topologies. Enero's code is publicly available [107].

Chapter 5

Network Traffic Compression

5.1 Introduction

To train the previously presented DRL agents we need to store large volumes of network data. For example, we could store the traffic evolution in a real-world network and then train the DRL agent to optimize the network configuration using realistic traffic matrices. The more data we use to train the DRL agents, the higher will be their performance when optimizing the limited network resources. This is because the training set will contain a wider range of different network scenarios, helping the DRL agent find better routing policies.

The NDT paradigm also requires the storage and analysis of vast amounts of network data [1]. This is because the ML models will learn to mimic the physical network's behavior from the collected data. Having large datasets improves the accuracy of the ML models used to mimic the network's behavior. In addition, to accurately model complex network behaviors such as end-to-end delay it is necessary to have large training datasets.

However, the efficient storage of network traffic traces is becoming more challenging than ever. Traffic traces from ISPs, backbone or data center networks can easily occupy hundreds of terabytes per day [110] or petabytes in the case of mobile networks [29]. For example, only a 24-hour trace from a single 10 Gbps link can result in 108 terabytes of data in the worst case. Storing traces from real-world networks can be difficult as they have in the order of hundreds of links [43]. In addition, such traces can contain thousands of concurrent flows per second [110]. Even storing aggregated flow-level information (e.g., NetFlow) can require hundreds of terabytes of disk storage per day [111, 112].

Traffic traces are being collected and compressed using generic methods such as GZIP [30]. However, such methods are generic, meaning that they were designed to compress multiple kinds of information (e.g., csv file, text). This results in low compression performances, resulting in traffic traces that have similar size than before being compressed. Past works showed that network traffic traces are far from being purely random, meaning that they

intrinsically have some underlying structure [32–36]. In particular, traffic traces are known to present spatial and temporal patterns that could potentially be exploited to increase current compression ratios.

In this chapter, we seek to understand if recent advancements in NNs architectures could be used to leverage spatial and temporal correlations to achieve better compression ratios than traditional tools. In particular, we present our second contribution of this dissertation which is a GNN-based traffic compression method. This method exploits spatio-temporal correlations naturally present in network traffic traces, achieving better compression ratios than traditional methods like GZIP.

The proposed compression method contains two main modules: a *predictor* that is implemented using neural networks and an *encoder*. The main role of the predictor is to exploit the spatial and temporal correlations between the network links to accurately estimate, from past observations, the distribution of the data to be compressed. The encoder is implemented using Arithmetic Coding (AC) [113], a popular lossless compression method. Based on the predicted distributions, AC decides how to better encode the traffic information. The proposed solution also implements a *decoder* for decompression, which inverts the process to recover the original traffic data.

To showcase the compression capabilities of our method, we first evaluate it on synthetically-generated traffic with different degrees of temporal and spatial correlation. The results with synthetic data show that our proposed solution can improve GZIP's compression ratios by ≥35%, even in scenarios with weak correlation. Next, we evaluate our compression method with real-world datasets that cover several months of traffic from three real-world networks. Experimental results show that our method can reduce the size of compressed files by 50%-65% compared to GZIP and by a factor between 2.6x and 4.2x with respect to the original file.

5.2 Background

In our work, we consider the compression scenario defined by a network topology with link-level traffic measurements. These measurements indicate the traffic volume over time going through each link. They can be obtained from the real-world network using network monitoring tools such as SNMP or NetFlow. Link-level measurements are performed periodically and stored in time bins (e.g., bins of 5 minutes), resulting in sequences of accumulated traffic values that we want to store efficiently in disk[1]. Figure 5.1 shows an overview of the compression scenario.

When compressing network traffic measurements, network administrators typically follow a simplistic approach based on well-known compression software such as GZIP [30].

[1]We chose this scenario for its relevance and simplicity, but note that the same principles apply for example to flow-level measurements (e.g., NetFlow), where flows (instead of link-level measurements) can be seen as multiple time series exhibiting spatio-temporal patterns.

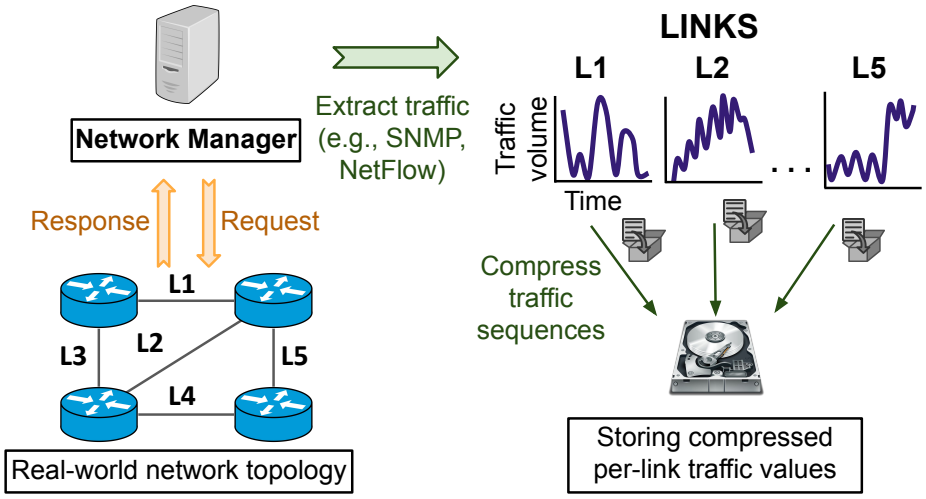


Figure 5.1: Overview of the network traffic compression scenario. Link-level traffic measurements are extracted from the real-world network and stored in the disk.

However, such methods are generic, meaning that they were designed to compress multiple kinds of information (e.g., images, text). This results in low compression ratios when used with network traffic traces. Equation 5.2.1 shows how to compute the compression ratio (CR) of a file.

$$CR = \frac{Uncompressed\_size}{Compressed\_size} \tag{5.2.1}$$

5.2.1 Exploiting temporal and spatial correlations

In our work, we leverage ML to exploit the network traffic characteristics and achieve high compression ratios. Specifically, link-level traffic measurements are time series data, meaning that the traffic values can be seen as a set indexed by time. A time series can be typically described by its seasonality and trend. Seasonality refers to a pattern repeated in time at a certain frequency (e.g., day/night). The trend indicates long-term tendency of the time series to increase, decrease or remain stable. Figure 5.2 shows the daily seasonality present in link-level traffic measurements during ≈1 month on two real-world datasets used in our experiments (see Section 5.4.1). In addition, the network topology and routing introduce spatial correlations in the link-level traffic measurements. This means that the links sharing paths are going to have a similar traffic behavior, which we believe that it can be exploited for improving the compression ratios.

To showcase the presence of spatial correlations, we compute the Pearson correlation coefficient between each pair of links in two real-world topologies. This coefficient indicates how strongly correlated are two sets of data or vectors. The following equation shows how to compute the Pearson correlation:

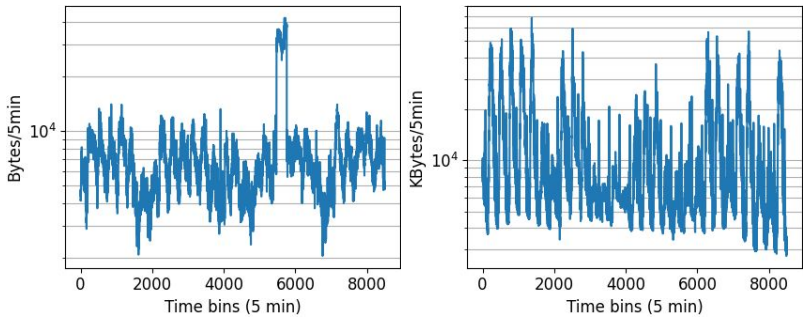


Figure 5.2: Two link-level network traffic measurements during ≈1 month from two real-world datasets. Notice the y-axis is in logarithmic scale. The figures indicate that the resulting time series from the measurements have temporal patterns.

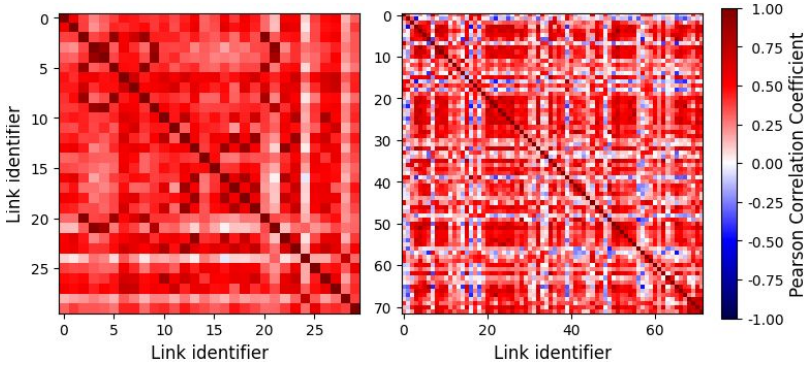


Figure 5.3: Pearson correlation between links for the Abilene (left) and Geant (right) datasets. The darker colors indicate high spatial correlation between links. This means that the traffic values between links have a positive correlation (red) or a negative one (blue).

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2(y_i - \overline{y})^2}} \tag{5.2.2}$$

where $\overline{x}$ and $\overline{y}$ are the means of the vectors x and y respectively. The resulting value is contained between the range [-1, +1], where -1 indicates negative correlation. This can happen when the traffic increases in one link but decreases in the other link. A value of 0 indicates no correlation and values close to +1 indicate positive correlation (i.e., the traffic increases in both links in similar proportions). Figure 5.3 shows the Pearson correlation for the real-world Abilene (left) and Geant (right) datasets [114]. The darker the color in the figure, the higher is the correlation between links. The figures indicate that indeed there is spatial correlation between links. We believe that both temporal and spatial correlations can be exploited to achieve higher compression ratios than generic methods like GZIP.

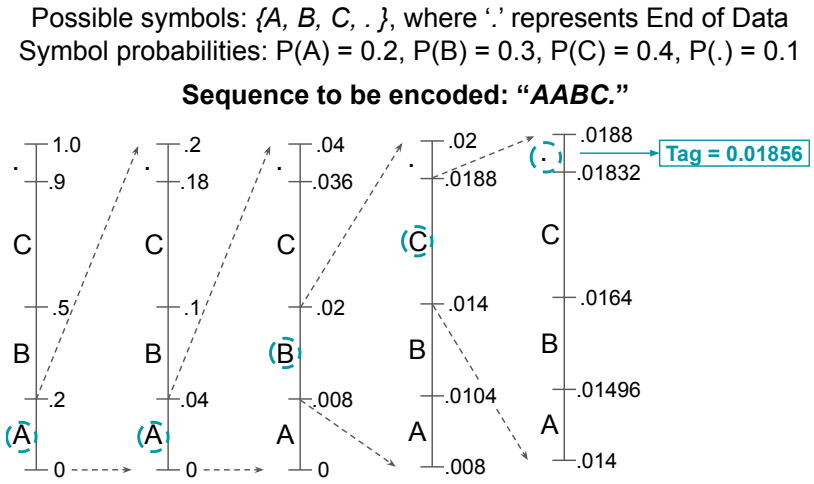


Figure 5.4: Given a finite set with all possible symbols and a probability distribution, the sequence "AABC" is encoded into a single decimal value. The process starts by dividing the range [0, 1) proportionally to the input distribution. Then, the process picks the segment that corresponds to the first symbol from the original sequence for further division. This process is repeated recursively until all symbols have been encoded.

5.2.2 Arithmetic Coding

Our method leverages AC [113] to compress the sequences of traffic values. This is a lossless method that compresses a stream of symbols (e.g., text characters) into a single number between [0, 1). To do this, AC assigns less bits to frequent symbols and more bits to less frequent symbols. In contrast to other popular compression methods such as Huffman coding [115], AC achieves better compression ratios and it can work in an online fashion. In addition, the AC compression algorithm works with probability distributions, making it a good fit with ML technologies.

Figure 5.4 shows the procedure to code a short text sequence using AC. Initially, the AC takes as input the set of possible symbols and a probability distribution. For simplicity, in this example the distribution remains static but a predictive model can be used to update the distribution after coding each symbol. Initially, the range [0, 1) is divided into segments proportionally to the symbol probability distribution. Then, the AC selects the segment [0, 0.2) corresponding to the first input symbol 'A' from the text sequence. Afterwards, this segment is divided into segments following the same proportions of the probability distribution. This process is repeated recursively for each symbol until the End-of-Data symbol is met. Finally, a decimal value from within the End-of-Data segment is picked as the tag for the entire text sequence.

The decoding part follows a symmetric procedure to the coding part. To recover the original text sequence, the algorithm takes as input the tag, the set of possible symbols and the probability distribution. Similarly, the range [0, 1) is divided into segments proportionally to the probability distribution and the segment that includes the codeword is selected. The symbol that corresponds to the segment represents the first symbol from the original

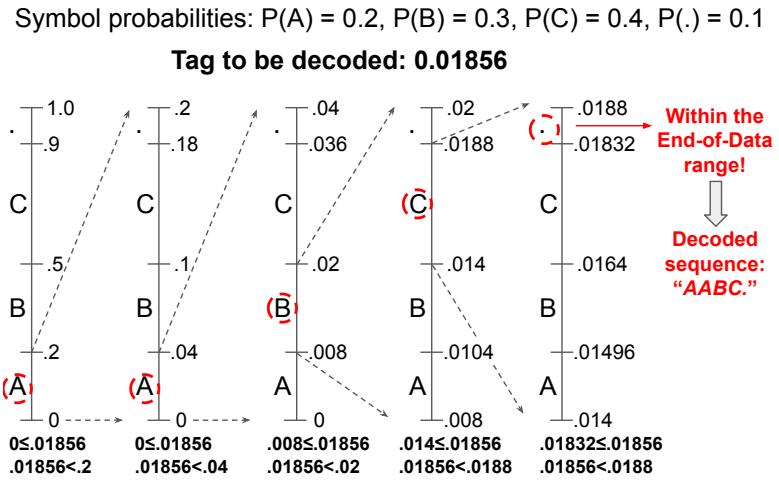


Figure 5.5: Given a tag, a set of symbols and a probability distribution, the decoding process starts by dividing the range [0, 1) proportionally to the distribution. The segment that contains the tag is selected and its corresponding symbol is decoded as the first symbol from the original text sequence. Then, the segment is divided proportionally, starting a recursive process that finishes with the End-of-Data symbol.

text sequence. Then, the process starts again, decoding the original sequence one symbol at a time. This is a recursive process that finishes when the End-of-Data symbol is met. Figure 5.5 shows an example of decoding the tag and recovering the original text message.

The compression performance of the AC is defined by the quality of the probability distribution. Consider a scenario where there is a predictive model to dynamically compute the probability distribution for each symbol in a sequence. As an example, consider the AC at time bin t and we want to compress the value at $t+1$. Then, the AC can use a predictive model that takes as input the past k symbols and predicts the probability distribution for the next symbol at $t+1$. If the model is accurate, AC will assign less bits to encode the symbol, resulting in close-to-optimal compression performance. On the other hand, if the model is not accurate, the probabilities will not correspond to the real symbol, which results in poor compression or it can even increase the final file size. In this work we want to leverage ML to build an accurate predictor to compress the sequences of network traffic measurements.

5.2.3 Notation and problem statement

Formally, link-level traffic measurements are represented as a matrix $\boldsymbol{X} \in \mathbb{N}^{w \times l}$, where w represents the sliding window for l links. Each traffic measurement is a random vector $\boldsymbol{x}_t \in \mathbb{N}^l$. For the arithmetic encoder we need a one step forecast distribution $p(\boldsymbol{x}_t|\boldsymbol{x}_{<t})$ to capture temporal dependence. As the arithmetic encoder operates on streams of symbols, we further partition the distribution with a chain rule to capture spatial dependence:

$$p(\boldsymbol{x}_t|\boldsymbol{x}_{<t}) = \prod_l p(x_l|\boldsymbol{x}_{t,<l}, \boldsymbol{x}_{<t}). \tag{5.2.3}$$

Here we assume the stationary model $p(x_l|\boldsymbol{x}_{t,<l}, \boldsymbol{x}_{<t})$ and mask-out the unknown traffic values. Note that there is no natural order for the auto-regressive model, however, the only requirement is that the order must be the same during compression and decompression.

5.3 Proposed Compression Method

In this section we present a method for network traffic compression based on NNs. This method compresses link-level traffic measurements that evolve during time. These measurements are performed periodically and aggregated in time bins. For example, the bins can be of 5 minutes, indicating the traffic that passed through a link during this period of time.

We consider two link-level compression scenarios. In the first one, we want to compress link-level traffic measurements from a single link (e.g., access link). This is a common practice in small or medium size networks where internal traffic is smaller and not considered to be of interest in many cases (e.g., enterprise, campus networks). With a single link, we can only exploit temporal correlations as no other links are considered. The second scenario represents a more general use-case, where we want to simultaneously compress the traffic from multiple links of a network topology (i.e., network-wide compression [116]). This situation will be more common in large networks, such as those of ISPs, which can have a global view of the network topology. In this case, apart from the temporal correlations of the first scenario, the routing and the topology will also introduce spatial correlations that we can exploit for compression purposes.

Our compression method takes as input the link-level traffic values and outputs a floating point number for each link. This value corresponds to the tag of the AC and it

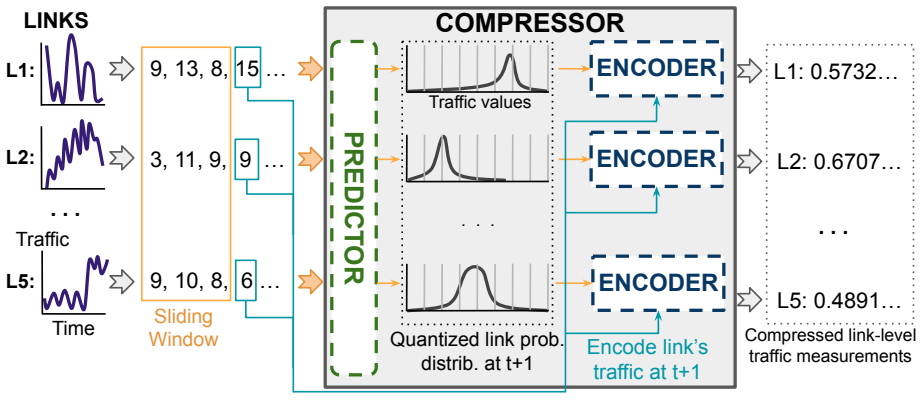


Figure 5.6: Our method takes as input the traffic values within the sliding window and outputs a tag per link. Suppose the window finishes at time bin t, the predictor computes the probability distribution of the traffic values at $t+1$. These are used to encode the real values from $t+1$. Afterwards, the sliding window is shifted by one time bin and the process starts again until the end of the sequence.

represents the entire sequence compressed (see Section 5.2.2). In the network-wide scenario (e.g., ISP), our proposed method takes as input the network topology in addition to the traffic values. The proposed compression method uses a sliding window to iterate over all traffic measurements. In other words, it processes the traffic values within the sliding window to output a probability distribution. This distribution is used to actually encode the traffic measurements immediately after the sliding window. The process is repeated until the window iterates over all values.

The proposed compression method is composed of two main blocks: the predictor and the encoder/decoder. The predictor leverages a NN model to predict the probability distribution in the next time bin after the sliding window (see Section 5.3.1). The encoder/decoder is in charge of actually compressing the sequences of traffic values (see Section 5.3.2). The better are the predictions of the NN model, the better is the compression ratio of our method. Figure 5.6 shows an overview of the compressor module, with its inputs and outputs.

5.3.1 Predictor

The predictor is implemented using a Recurrent Neural Network (RNN) in its simplest form. This implementation is used when compressing link-level traffic measurements of a single link. Specifically, the RNN processes the link-level sequence of traffic values and afterwards a Multi-Layer Perceptron (MLP) takes the resulting hidden states and outputs the parameters of a probability distribution (e.g., Normal, Laplace). The probability distribution is then used by the AC to code the real link measurements. Notice that the RNN architecture only exploits temporal correlations.

In the network-wide scenario, we implement the predictor using a Spatio-Temporal Graph Neural Network (ST-GNN) [117]. Inspired by the message passing neural network [68], the proposed ST-GNN uses a message passing step for each time bin to exploit the spatial and temporal correlations. This step consists of exchanging information between neighboring links and it is necessary to propagate the link-level information across the topol-

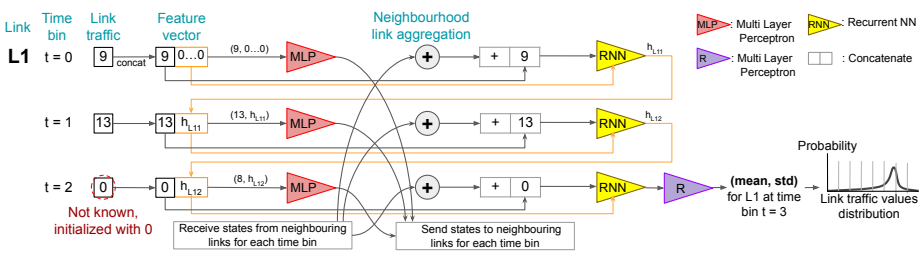


Figure 5.7: The initial feature vectors are processed by a MLP. Then, for each link the hidden states of its neighbours are aggregated and concatenated with the actual link utilization. The resulting hidden state is processed by a RNN, which outputs a new state used to initialize the same link feature vector in the next time bin. Finally, a different MLP outputs the mean and standard deviation of a probability distribution.

ogy. The ST-GNN takes as input the traffic measurements and the network topology and outputs a per-link probability distribution. The ST-GNN enables to exploit both spatial and temporal correlations naturally present in network traffic traces (see Section 5.2.1).

Figure 5.7 shows an overview of the internals of the ST-GNN architecture. For simplicity, we only show the steps of predicting the probability distribution in a single link *L1* using a time window of size 2. Initially, at time bin *t=0* the links' feature vectors are initialized with the traffic values and padded with 0. Then, the feature vector is processed by a MLP and the output vector is sent to all the neighboring links. At the same time, the current link *L1* receives the hidden states from the neighboring links, aggregates them using a sum and concatenates the actual link traffic value. The resulting hidden state is then processed by the RNN, which outputs a final hidden state for the present time bin. This hidden state is used the next time bin *t=1* to initialize the feature vector. The process is repeated for all time bins within the sliding window. In the last bin, a different MLP (denoted R in the figure) is used to process the final hidden states and to output the mean and standard deviation of the traffic value probability distribution (e.g., Normal, Laplace). This probability distribution is used by the AC to compress the traffic value of link *L1* at time bin *t=2*.

5.3.2 Encoder/Decoder

The encoder/decoder is responsible for effectively compressing/decompressing the actual sequences of traffic values. Specifically, it takes the link-level probability distributions from the predictor and compresses/decompresses the sequences of traffic values. We implement the encoder/decoder using AC [113], which compresses each link-level traffic sequence into a single floating point number (i.e., one decimal number per link). When decoding, the method works symmetrically to the encoder (see Section 5.2.2). The more accurate the NN-based predictor, the higher the compression ratios as less bits will be used for compressing the traffic sequences.

5.3.3 Mask

The ST-GNN model uses a mask to exploit the spatial correlations between links, enabling the model to learn the conditional distribution $p(x_l|\boldsymbol{x}_{t,<l}, \boldsymbol{x}_{<t})$ from Equation 5.2.3. Specifically, the mask is used to gradually incorporate the already compressed/decompressed link traffic values of a time bin into the prediction. By masking the known link traffic values, our model learns to predict the conditional probability distribution. As an example, consider a topology with 2 links and a time window of size 2. This means we know all the link traffic values for time bins *t=0* and *t=1*. Then, the ST-GNN uses the known traffic values to predict the probability distributions for both links at *t=2*. From all the distributions, a single one is picked and the corresponding link traffic value is compressed using AC. The link is then marked as known for bin *t=2* using the mask and the ST-GNN uses the updated

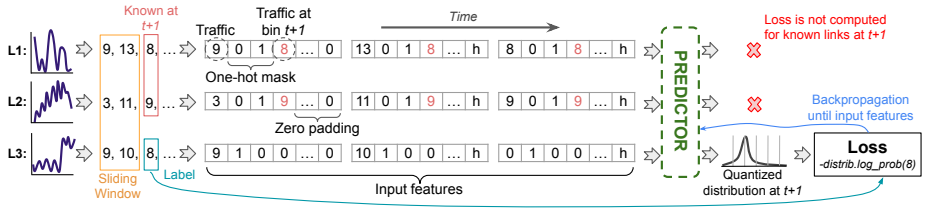


Figure 5.8: Consider a time window of size 2 that finishes at time bin t. From time bin $t+1$, we already compressed 2 traffic values from link $L1$ and $L2$ and we want to compress the value in $L3$. The feature vectors assigned to each link for each time bin are initialized with the known values and the mask. This information is processed by the predictor, which outputs a quantized probability distribution for the missing link value at $t+1$. The loss is computed for the same link and back propagated all the way to the link input features.

link features to predict the probability distribution for the missing link. This prediction is conditional to the known traffic value, and thus exploiting the spatial correlation between links.

During training, the mask of unknown links is created randomly. This means that for each sliding window we associate a random mask over the links to indicate whose link traffic values are known. In the compression/decompression phase, the mask starts by marking all the traffic values as unknown. Then, the predictor compresses/decompresses the traffic values in order, incorporating them into the prediction by changing the mask. Figure 5.8 illustrates how the link-level features are initialized and how the loss is computed for a single link. In particular, for each link and time bin, the input link features are the traffic measurements and the mask.

5.3.4 Compression

Our proposal uses the link-level traffic values from the sliding window to predict the probability distributions, including the masked links from the next time bin. In particular,

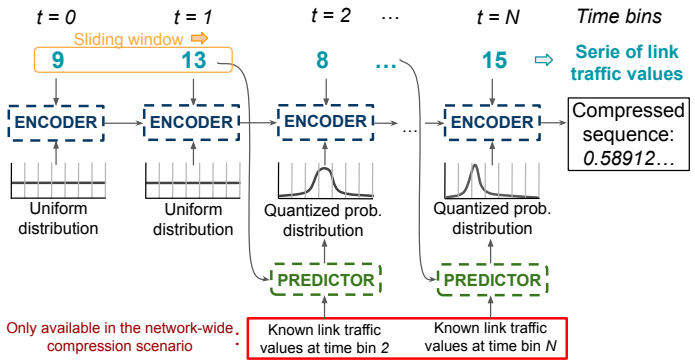


Figure 5.9: The predictor processes the information from the sliding window. In the network-wide scenario, the predictor incorporates information from the links whose traffic is known at $t+1$ to predict the conditional probability distribution $p(x_l|\boldsymbol{x}_{t,<l}, \boldsymbol{x}_{<t})$. The output is a single decimal value that encodes the link's traffic sequence.

Algorithm 5 Compression algorithm

```
1:  Inputs: seq_windows                                  ▷ Sequence of ordered windows
2:  first_w ← True
3:  for each w ∈ seq_windows do
4:      if first_w then
5:          prob_dist ← uniform_dist
6:          for each bin ∈ range(len(w) − 1) do
7:              for each link ∈ range(num_links) do
8:                  encode(w[link, bin], prob_dist)
9:          first_w, bin ← False, len(w)
10:     for each l ∈ range(num_links) do
11:         prob_dist ← model_inference(w)
12:         prob_dist ← quantize(prob_dist)
13:         if scenario == network − wide then
14:             link ← select_link(prob_dist)
15:             encode(w[link, bin], prob_dist)
16:             w ← update_link_features(w, link)
17:         else
18:             link ← l
19:             encode(w[link, bin], prob_dist)
```

if the time window finishes at time bin t, it uses the previous k traffic values until t to predict the probability distributions for time bin *t+1*. These distributions are then used by the AC to code the actual values from time bin *t+1*. Figure 5.9 shows an overview of the compression process for a single link. In the network-wide compression scenario, the predictor is implemented with a ST-GNN that takes as additional input features the neighboring link's hidden states. When compressing a single link, the predictor is implemented with a RNN that has only information from the past traffic values independently for each link.

Algorithm 5 shows in detail how the compression procedure works. To simplify the pseudocode, we compress the traffic values in the last time bin from a sliding window. The algorithm takes as input the ordered sequence of time windows and starts iterating over them (line 3). The first traffic values from the first time window are compressed using uniform probabilities (line 5 to line 8). Then, the algorithm encodes the traffic values from the last position of the time window (line 9). To do this, a loop iterates over each link, compressing one value at a time (line 10). For each link, the algorithm leverages the NN-based model to compute the quantized probability distributions (lines 11 and 12).

The quantization step is necessary because the AC takes as input a probability distribution with a finite set of values (see Section 5.2.2). When working with high traffic granularities (e.g., store the traffic values until the last byte), the total number of different traffic values can become very large and sometimes the probability distribution might not fit in memory. In these cases, a solution would be to quantize the traffic values to the Kilobytes or Megabytes, instead of quantizing at the byte-level. This process can be seen

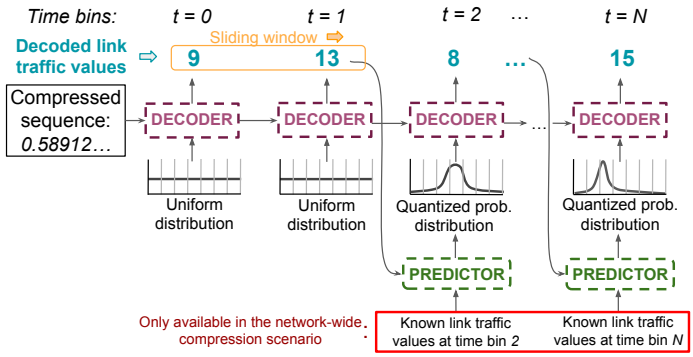


Figure 5.10: The decompression process is similar to the compression, but now the arithmetic coding uses the decoder to recover the original sequence of symbols. Notice that for each time bin the predictor receives the same input information as in the compression phase.

as grouping the range of traffic values in larger bins of Kilobytes or Megabytes and the probability distribution is produced over these larger bins. The binning step produces some loss of information as we loose precision at the byte granularity, but in some cases it is necessary due to hardware limitations.

The model that outputs the probability distribution is implemented with a RNN or a ST-GNN, depending on the compression scenario. In the network-wide scenario, the algorithm uses a heuristic to determine the link order (line 14), incorporating them into the prediction. After encoding the selected link, the link-level features and the mask are updated in line 16. The process starts again and is repeated until all links have been encoded.

The heuristic to select the link in the network-wide scenario is based on an increasing order of standard deviation (line 14). We experimentally observed this heuristic helps the ST-GNN decrease the uncertainty in the predictions. In other words, leaving the links where the GNN model is more uncertain to the end helps the GNN make better predictions. This is because the ST-GNN will have more links with known traffic values, reducing the model's uncertainty over the links with higher standard deviation.

The compression process results in a tag for each link, encoding the link's traffic sequence. This tag is stored in a single file on disk. When decompressing, the tag is used by the decoder to recover the original traffic sequence without losing information. Notice that the compression process is performed in a streaming fashion, differing from traditional methods like GZIP that are static. In other words, our method can compress the traffic values as they come, without the need of storing the measurements in a buffer before compressing.

5.3.5 Decompression

For decompression, the model reads the tag from the compressed files and uses the same NN-based model to recover the original link sequences bin by bin. The first elements within the first sliding window are decompressed using uniform probability distributions. Then, when the entire sliding window is decompressed, the predictor uses the recovered values to compute the probability distributions for the links. Similarly, in the network-wide compression scenario the algorithm leverages the same heuristic to select the order in which to decode the traffic values for each time bin. Figure 5.10 shows a general overview of the decompression phase. Notice that for each time bin, the predictor receives the same input information as in the compression phase. The decoder also receives the same information but in this case its operations are inverted to decode (see Section 5.2.2). The algorithm to decompress is the same as Algorithm 5 but replacing the coding operations from *encode(·)* by their inverse decoding operations.

5.4 Experimental Evaluation

5.4.1 Methodology

We evaluated the compression performance of our method with respect to GZIP, the *de facto* compression method of network traffic traces. In the first experiment, we generated synthetic datasets with different intensities of spatial and temporal correlations on the NSFNet topology [88] with 42 directional links. We synthetically created signals with different correlations (see Section 5.4.3) and extracted 1,000 samples using a window size of 5 time bins, including the labels. In other words, the NN-based models leverage the link-level traffic values of 4 time bins to compress the values within the 5th bin. During training, we created 50 different random masks for each time window. We experimentally observed that the higher the number of different masks, the better is the accuracy of the ST-GNN model, but paying the cost of increasing training time. This is because increasing the number of different masks enriches the training process as there is more data to train the NN on.

In the second experiment, we evaluated the compression capabilities of the NN-based models on three real-world datasets. The first two datasets are the Abilene and Geant datasets from [114]. The Abilene dataset corresponds to a topology with 30 directional links and a total of 41,741 samples after data cleaning and using a time window of size 5. This dataset contains the link-level traffic measurements (in bytes) during 6 months in intervals of 5 minutes. The Geant dataset corresponds to a topology of 72 directional links and a total of 6,063 samples after data cleaning for the same window size. The third dataset is more recent and it was obtained from in-house link-level traffic measurements in a campus network. The dataset contains 12 months of per-link network traffic measurements

in intervals of 5 minutes (from December 2020 until December of 2021). The topology contains 16 directional links and a total of 102,799 samples with window size of 5.

All the experiments were performed on *off-the-shelf* hardware. In particular, we used a single machine with an AMD Ryzen 9 5950X 16-Core Processor with one GeForce GTX 1080 Ti GPU for training the models. We trained all NN-based models using 70% of the samples for training and 30% for evaluation. For each time bin, we created 40, 40 and 20 unique random masks for the Abilene, Geant and Campus network datasets. The loss function used was the negative log likelihood of a Laplace distribution. To work with a finite set of probabilities, we quantized the Laplace distribution. After training, we chose the model with lowest evaluation error and we compressed the entire datasets.

5.4.2 Implementation

We implemented the ST-GNN and the RNN using Tensorflow 2.8 [85]. The RNN was implemented using the Gated Recurrent Unit architecture [118]. The scripts to pre-process the datasets were written in Python 3.8 and we used the NetworkX [92] and NumPy [105] libraries for graph-related operations. We leveraged an open-source implementation of the arithmetic coding for Python [119] to implement the encoder. In the synthetic experiment, we used the Statsmodels Python library [120] to implement the auto-regressive model.

5.4.3 Synthetic data generation

In our compression scenario, we assumed the network topology is an input to the model (only in a network-wide scenario). The only remaining variables that have an impact on the link-level traffic measurements are the source-destination flows. There is one flow for each pair of source-destination nodes within a network topology. All flows follow the shortest path routing policy to reach the destination node. Consequently, each link will be traversed by a subset of all flows.

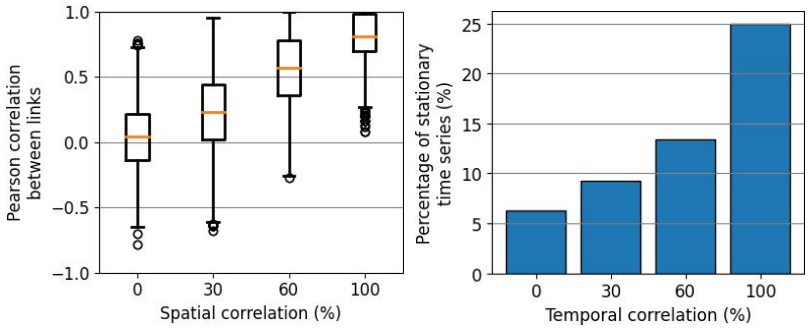


Figure 5.11: Boxplots of the Pearson correlation in the synthetic datasets (left). The higher is the spatial correlation, the higher are the pearson correlation coefficients between links. On the right, we show the higher is the temporal correlation, the higher is the percentage of link-level stationary time series in the synthetic datasets.

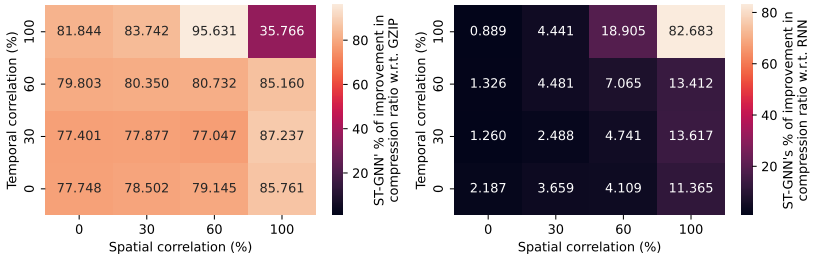


Figure 5.12: Compression ratio improvement for the ST-GNN model with respect to GZIP (left) and RNN (right). Notice that in the scenario with higher spatial and temporal correlations there are a few traffic values that are highly repeated in the dataset, which GZIP's underlying algorithm exploits effectively.

We assigned for each flow a periodic signal that originated from a *sine* wave. This wave is scaled by 40 to obtain values in the order of hundreds when aggregated on each link and is shifted to contain only positive values. In addition, we add random noise to the signal, we randomly shift its phase to start at different values and we randomly change the periodicity. If all flows originate from the same signal, the links are highly correlated in space as their values will increase/decrease proportionally for each time bin. In our experiment, we consider 4 degrees of spatial correlation: 0%, 30%, 60% and 100%, indicating the percentage of flows that have the same signal characteristics.

Temporal correlations are present in stationary time series with periodical patterns. The *sinusoidal* signal naturally meets these requirements, resulting in a time series with high temporal correlation. To decrease the temporal correlation, we added additional noise to the flow signal from auto-regressive (AR) models. To control the intensity of the temporal correlation in the experiment we controlled the percentage of flows that are added with noise. Specifically, we consider 4 degrees of temporal correlations: 0%, 30%, 60% and 100%, indicating the percentage of flows that conserve the original signal.

In Figure 5.11 (left) we show how the spatial correlation increases in our synthetic datasets. Specifically, we grouped all the datasets by the intensity of spatial correlation. Then, we computed the Pearson correlation for each pair of links following Equation 5.2.1. The results indicate that the higher the intensity of the spatial correlation (x-axis), the higher the Pearson correlation (y-axis). Figure 5.11 (right) shows how the temporal correlation present in the synthetic data increases with the temporal correlation coefficient (x-axis). Similarly, we grouped all datasets by temporal intensity. Then, we performed the Augmented Dickey-Fuller (ADF) test [121] for each link-level traffic sequence, counting the number of paths that are stationary (i.e., the mean and variance of the time series do not change in time). In particular, if the ADF test was returning a *p-value* smaller or equal than 0.05, we rejected the null hypothesis (H0), considering the time series to be stationary.

5.4.4 Evaluation on synthetic data

In total, there are 16 experiments that correspond to all possible combinations of spatial and temporal correlation intensities. For each of these experiments, we consider both network-wide and independent link-level compression scenarios (see Section 4.3). In total, we trained 32 models from which 16 of them were based on the ST-GNN model and the other 16 on RNNs.

Figure 5.12 (left) shows the percentages of compression ratio improvement for the ST-GNN with respect to the GZIP baseline in the network-wide scenario. The results indicate that the ST-GNN outperforms GZIP by a large margin in all experiments. Notice that the scenario with maximum spatial and temporal correlation contains a small number of link-level traffic values that are repeated frequently. GZIP uses Huffman coding [115] as the underlying algorithm, which can effectively exploit the repeated traffic values. This explains why the compression ratio improvement is the lowest for this particular scenario. In addition, the figure indicates the expected performance of the ST-GNN when evaluated in real-world scenarios. In particular, the intensity of the temporal and spatial correlations of a real-world dataset could point to the expected performance with respect to GZIP.

To showcase the capabilities of our method to exploit spatial and temporal correlations simultaneously, we compare it with the RNN-based model. Recall that the RNN compresses one link only. Therefore, we apply the RNN model for each link in the topology, exploiting temporal correlations solely. For the sake of fairness, we maintained the same hidden state sizes in both ST-GNN and RNN models.

Figure 5.12 (right) shows the performance improvement of the ST-GNN models with respect to the RNN-based models. The ST-GNN model outperforms the RNN in all correlation scenarios, but it has outstanding performance in scenarios with high spatial correlation. The results indicate that our model has the flexibility to exploit both spatial and temporal correlations. Notice that in the case of 0% of spatial correlation and maximum temporal correlation, the improvement of the GNN model is $\approx$1%, indicating that they perform similarly when there is high temporal correlation. This is expected as the ST-GNN model also incorporates a RNN (see Figure 5.7).

5.4.5 Compressing real-world data

In this experiment we evaluated the compression performance of our method on real-world link-level traffic measurements. To do this, we compressed three real-world datasets (see Section 5.4.1). We compared the compression performance against three baselines: Static AC, Adaptive AC and GZIP. The Static AC and Adaptive AC baselines are similar to our method but the probability distribution is computed without using ML. In particular, Static AC iterates over the entire dataset and computes the probability distribution for the link-level traffic measurements. Then, it compresses/decompresses the entire dataset using the same static distribution for each AC coding step. The Adaptive AC baseline

computes the new distribution using the values within the sliding window. These baselines are intended to show the benefit of using ML to implement the predictor model. Finally, we apply GZIP to compress the entire dataset.

Figure 5.13 shows the compression ratios for the three real-world datasets in the link-level scenario. In particular, each baseline was applied to compress each link individually and we compared the resulting compressed links with their original file size (i.e., one file per link). The results indicate a remarkable performance of our compression method for all datasets, outperforming GZIP by a large margin. In addition, the figure indicates a clear advantage of using an adaptive ML-based predictor to exploit temporal correlations present within the sliding window.

Figure 5.14 shows the experimental results of compressing the same datasets in the network-wide scenario. Recall that our compression method is implemented using the ST-GNN, which leverages the traffic values from all links to dynamically compute the probability distributions. In this scenario, Static AC computes the probability distribution using the entire dataset (i.e., including all links) and Adaptive AC updates the distribution by including the values from all links within the sliding window.

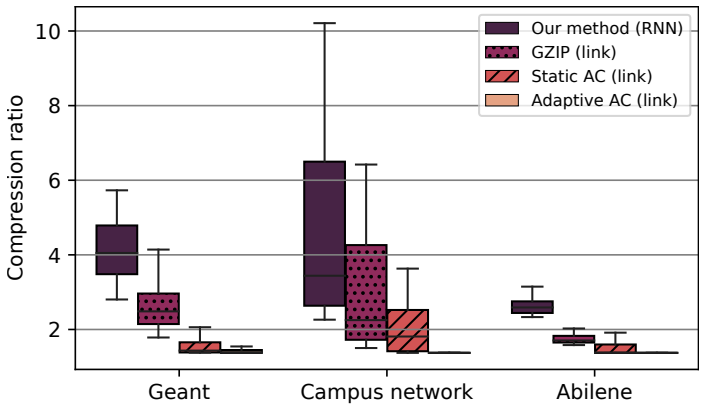


Figure 5.13: Compression ratios for the single-link scenario.

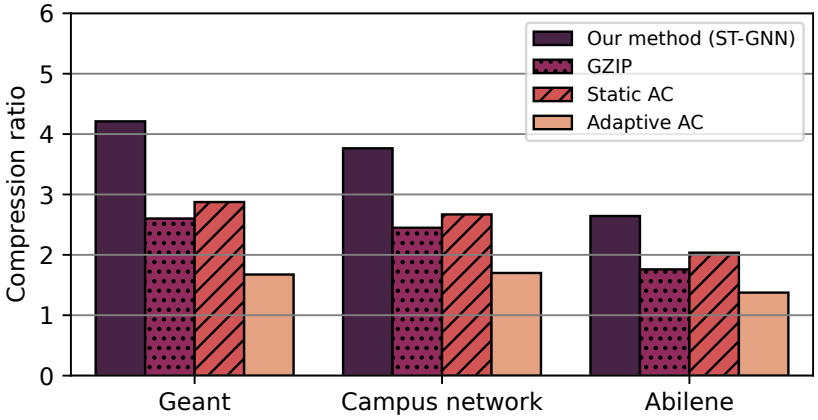


Figure 5.14: Compression ratios for the network-wide scenarios. Notice that in this experiment we are compressing the entire dataset.

Dataset	Mean Cost (s)		Model size (Kbytes)	
	ST-GNN	RNN	ST-GNN	RNN
Geant	3.20	0.47	575	489
Campus network	0.22	0.07	236	202
Abilene	1.22	0.29	332	285

Table 5.1: Model size and mean cost (in seconds) to compress one time bin.

The experimental results indicate that our proposed method achieves the highest compression ratios in all three datasets. Particularly, it outperforms Static AC by ≈47%, ≈41% and ≈29% for the Geant, Campus network and Abilene datasets respectively. In addition, the performance improvement with respect to GZIP is of ≈62%, ≈53% and ≈50% for the same datasets respectively. These results showcase the benefit of leveraging ML to exploit spatial and temporal correlations for traffic compression.

5.4.6 Cost

In this section we discuss the cost of using our method for online compression. Specifically, our method compresses the traffic measurements in a streaming fashion (see Section 4.3). This means that it can compress the link traffic measurements as they come from the network monitoring platform. Conversely, GZIP needs to wait to have the entire dataset to apply the compression algorithm. Alternatively, GZIP could compress all links at once on each time bin independently. We did this and the experimental results indicate that GZIP achieves a compression ratio of ≈0.94, ≈0.4 and ≈0.54 for the Geant, Campus network and Abilene datasets. In other words, the compressed data occupies more space than the original data, contrasting with the results from Figure 5.14.

We computed the average cost of compressing one time bin using the ST-GNN and RNN models to evaluate the deployment to real-world online traffic compression. In addition, we computed the size of storing the model's weights into a file. Table 5.1 shows the average cost of compressing one time bin for all real-world datasets, indicating that our method is capable of online compression. In other words, when it receives the aggregated traffic of, for example, the last 5 minutes, our method can effectively compress the values in the order of seconds. Finally, Table 5.1 also shows how much memory it is required to store the trained weights of the neural network. The results indicate that the model is lightweight and it achieves high compression ratios with an expendable model size.

5.5 Chapter Contributions

In this chapter we proposed the use of GNNs and arithmetic coding to compress link-level traffic measurements. Specifically, we presented a method that exploits the spatial and temporal correlations intrinsic in the traffic measurements. The experimental results

show that the proposed solution can effectively compress real-world traffic traces, with an improvement of ≥50% in compression ratio for real-world datasets with respect to GZIP.

Chapter 6

Related Work

6.1 Network Optimization

Network optimization is a well-known and established topic whose fundamental goal is to operate networks efficiently. This problem has been largely studied in the past and we outline some of the most relevant works. The early work from [41] uses LS to find the best OSPF link weights to minimize congestion in the most congested link. The authors from [39] propose a solution based on ILP for multicast routing in OTN. In addition, they propose to use a heuristic based on genetic algorithms to improve the locally optimal solution and to reduce the computational complexity. Similar work using genetic algorithms are [122, 123]. In DEFO [42] they propose a solution that converts high-level optimization goals, indicated by the network operator, into specific routing configurations using CP. Their problem formulation leverages SR to find the best routing configuration for each traffic demand. Within a SR path, they spread the traffic among several flows using ECMP. In [124], the authors propose to use LS where they sacrifice space exploration to achieve lower execution times. In their design they also leverage heuristics to narrow down the LS neighborhood and to make the algorithm converge faster to good solutions. A more recent work [40] leverages the ILP and the column generation algorithm to solve TE problems. Their solution also provides a mathematical bound to indicate how far the solution is from the optimal one.

6.1.1 Machine Learning for Network Optimization

To find the optimal routing configuration from a given traffic matrix is a fundamental problem, which has been demonstrated to be NP-hard [42, 125, 126]. In this context, several DRL-based solutions have been proposed to address routing optimization. In [55] they use a DRL solution for spectrum assignment using Q-learning and convolutional neural networks. Similarly, in [56] they design a more elaborated representation of the network state to help the DRL agent capture easily the singularities of the network topology. In [127] the authors

design a scalable method to solve TE problems with DRL in large networks. The work from [100] combines DRL with Linear Programming to minimize the utilization of the most congested link. In [128] they propose and compare different DRL-based algorithms to solve a TE problem in SD-WAN.

However, most of the proposed DRL-based solutions fail to generalize to unseen scenarios. This is because they pre-process the data from the network state and present it in the form of fixed-size matrices (e.g., adjacency matrix of a network topology). Then, these representations are processed by traditional neural network architectures (e.g., fully connected, convolutional neural networks). These neural architectures are not suited to learn and generalize over data that is inherently structured as a graph. Consequently, SoA DRL agents perform poorly when they are evaluated in different topologies that were not seen during the training.

There have been several attempts to use GNNs in the communication networks field. In [129] they use GNNs to learn shortest-path routing and max-min routing in a supervised learning approach. In [130] they combine GNNs with DRL to solve a network planning problem. Another relevant work is the one from [131] where they use a distributed setup of DRL agents to solve a TE problem in a decentralized way. The work from [79] proposes to use GNNs to predict network metrics and a traditional optimizer to find the routing that minimizes some network metrics (e.g., average delay). Finally, GNNs have been proposed to learn job scheduling policies in a data-center scenario without human intervention [132].

6.2 Machine Learning for Network Traffic Compression

The most widely used method for network traffic compression is GZIP. However, the networking community has investigated different approaches for compressing network traffic. The work of [133] proposes a lossy method to approximate the real network traffic by capturing the most relevant traffic features. In [134] they propose to exploit the traffic redundancies at the packet level to reduce the transmitted traffic. In [135] they propose an architecture to implement the LZ77 compression algorithm [136] on a FPGA. The work of [137] describes a solution for on-the-fly storage, indexing and querying of network flow data. A more recent work leverages the P4 language [138] and generalized deduplication to implement a solution that operates at line-speed.

The compression method presented in this dissertation has similarities with the problem of traffic prediction. The works of [139,140] propose the use of RNNs to predict the network traffic. A more recent work proposes to use simulated annealing and an Autoregressive Integrated Moving Average model [141] to predict the network traffic. In [142] they use a graph-based ML algorithm to predict the link-level traffic loads in backbone networks. The work from [35] they leverage inter-flow correlations and intra-flow dependencies to predict the traffic matrix using RNNs. Finally, the work from [143] proposes to use a

spatio-temporal convolutional neural network with attention mechanisms to predict wireless traffic.

Despite the similarities with traffic prediction, it is important to remark that the traffic compression problem addressed in our work has some particularities. First, our compression method only considers the probability distribution for each link, instead of the exact traffic values. This is due to the requirements of the arithmetic coding part. Second, we only work with the values from the next time bin, whereas in traffic prediction the work horizon is typically larger (e.g., predict the traffic for the next couple of hours). Finally, when decompressing information we do not have access to the first elements of the time-window, forcing the use of simple methods that do not depend on the compressed information (e.g., uniform probability).

Chapter 7

Conclusions and Future Work

In the last years, the NDT paradigm emerged as a key enabler for efficient control and management of modern communication networks. A digital twin of a network can be seen as a virtual representation of a network that accurately mimics the real-world network behavior. One of its most important benefits is that it can model complex systems at a smaller cost. This contrasts with network simulators (e.g., OMNet++) that are computationally intensive, making them unsuitable for real-time network optimization. There are other analytical models that are less resource demanding such as queueing theory. However, these techniques make strong assumptions about the network scenario and their performance can be far from being accurate.

This dissertation visited the NDT paradigm to design new optimization methods suitable for real-time network operation. The first contribution of this thesis is a DRL architecture based on GNNs that is able to maximize the network resources utilization. The experimental results showed that the proposed DRL+GNN architecture enables efficient network operation on different networks than those used for training. This is key for real-time network management where changes happen frequently.

To show the generalization capabilities of our DRL+GNN solution, we selected a classic problem in the field of optical networks. This allowed us to have a baseline benchmark to validate the performance and generalization capabilities of the proposed architecture with respect to SoA methods. The experimental results showed that the DRL+GNN architecture is able to effectively operate in networks never seen during training. The source code and the experimental results are publicly available and can be found in [80].

To further explore the capabilities of the DRL+GNN architecture, we applied it to a more complex and realistic optimization scenario in IP networks. Specifically, the second scenario starts with a default routing configuration and the DRL agent needs to minimize the utilization of the most loaded link by changing the routing. The DRl+GNN architecture was improved by implementing a new learning algorithm, designing a new action space and changing the training methodology to incorporate data from multiple network scenarios. In

addition, we combined the ML model with a local search heuristic to improve the DRL's solution. The experimental results showed that the proposed method is able to operate efficiently in real-world scenarios in real-time. In addition, the results indicated that the proposed solution achieves close-to-optimal performance in less than 30 seconds for a set of arbitrary real-world network topologies. The source code and the experimental results can be found in [107].

The optimization results from both scenarios validated the DRL+GNN architecture for network optimization. Network operators can use the proposed architecture to improve their real-time network performance without the need of retraining the DRL agent when there is a link failure or the traffic matrix changes. This architecture can be easily extended by integrating advanced NDTs to optimize more complex metrics, ensuring SLAs for modern network applications. In addition, the generalization capabilities enable a new breed of networking products. For example, network operators can train a DRL agent in a controlled lab (for instance at the vendor's facilities) and ship it to the customer. Once deployed, the DRL agent can efficiently operate the unseen network or scenario of the customer.

Storing network related data is becoming more challenging as the network size grows in traffic volume and network users. This data is of the utmost importance to train accurate ML models that are used to build NDTs or to train DRL agents for network optimization. In this dissertation we proposed a new compression method that uses ML-based models and arithmetic coding to compress link-level traffic measurements. The ML models were designed to exploit the spatial and temporal correlations intrinsic in the network traffic traces. The experimental results showed an improvement of $\geq$50% in compression ratio for real-world datasets with respect to GZIP.

Our compression method can benefit network operators to store network-related time-series data more efficiently. Generic out-of-the-box NN-based models can be commercialized as plug and play compression methods that are easily installed using widely known package-management systems. Specifically, the generalization capabilities of GNNs enable these models to efficiently compress data from other graphs without the need of retraining.

7.1 Future Work

A fundamental challenge that remains to be addressed towards the deployment of DRL techniques to enable NDT-based optimization is their black-box nature. In particular, DRL does not provide guaranteed performance for all network scenarios and its operation cannot be understood easily by humans. As a result, DRL-based solutions are inherently complex to troubleshoot and debug by network operators. In contrast, computer networks have been built around well-understood analytical and heuristic techniques, based on well-known assumptions that perform reasonably well across different scenarios. However, such

issues are not unique to ML-based network optimization, but they are rather common to the application of machine learning to many critical use-cases, such as self-driving cars.

In our work we combined DRL with a local search heuristic to mitigate the poor performance of the DRL agent in out of distribution data. This was done to improve DRL's performance in case it got stuck in a local minima solution when the network optimization scenario is very different from those seen during training. As future work, the DRL agent performance could be further improved by adding a wider range of network scenarios in the training set. In addition, the combination of DRL with more sophisticated solvers (e.g., CP, ILP) or heuristics could be explored. We believe that it would help achieve better performance at a small execution overhead.

Finally, our work could be extended to enable optimization with flow-based NDTs for fine-grained control and management. Having small granularity control is relevant for network operators to achieve more flexible policies. However, this challenging because there are in the order of hundreds of thousands of network flows that have a very short periods of times before they finish. Consequently, optimization algorithms should be fast enough to optimize before the flows finish.

Appendices

Appendix A

List of publications

A.1 Related Publications

Journal Papers:

- Paul Almasan, José Suárez-Varela, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case." ***Computer Communications (JCR Q2)***, 196, pp. 184-194, 2022.
- Paul Almasan, Xiao Shihan, Xiangle Cheng, Xiang Shi, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "ENERO: Efficient real-time WAN routing optimization with Deep Reinforcement Learning." ***Computer Networks (JCR Q1)***, vol. 214, 2022.
- Paul Almasan, Miquel Ferriol-Galmés, et al., "Network Digital Twin: Context, Enabling Technologies and Opportunities," in ***IEEE Communications Magazine (JCR Q1)***, doi: 10.1109/MCOM.001.2200012.

Conference Papers:

- Paul Almasan, José Suárez-Varela, et al. "Towards real-time routing optimization with deep reinforcement learning: Open challenges." In 2021 IEEE 22nd International Conference on High Performance Switching and Routing ***(HPSR)***, pp. 1-6. IEEE, 2021.

A.2 Other Publications

- **(under review)** Paul Almasan, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangle Cheng, Albert Cabellos-Aparicio, and Pere Barlet-Ros. "Atom: Neural Traffic Compression with Spatio-Temporal Graph Neural Networks."

Journal papers:

- Krzysztof Rusek, José Suárez-Varela, Paul Almasan, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "RouteNet: Leveraging Graph Neural Networks for network modeling and optimization in SDN." IEEE Journal on Selected Areas in Communications ***(JSAC)*** 38, no. 10 (2020): 2260-2270.
- José Suárez-Varela, Paul Almasan et al., "Graph Neural Networks for Communication Networks: Context, Use Cases and Opportunities," in ***IEEE Network***, doi: 10.1109/MNET.123.2100773.
- José Suárez-Varela, Miquel Ferriol-Galmés, Albert López, Paul Almasan, Guillermo Bernárdez, David Pujol-Perich, Krzysztof Rusek, Loïck Bonniot, Christoph Neumann, François Schnitzler, François Taïani, Martin Happ, Christian Maier, Jia Lei Du, Matthias Herlich, Peter Dorfinger, Nick Vincent Hainke, Stefan Venz, Johannes Wegener, Henrike Wissing, Bo Wu, Shihan Xiao, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2021. "The graph neural networking challenge: a worldwide competition for education in AI/ML for networks." SIGCOMM Computer Communications Review ***(CCR)***, 51, 3 (July 2021), 9–16. https://doi.org/10.1145/3477482.3477485
- Davide Careglio, Salvatore Spadaro, Albert Cabellos, Jose Antonio Lazaro, Pere Barlet-Ros, Joan Manel Gené, Jordi Perelló et al. "Results and Achievements of the ALLIANCE Project: New Network Solutions for 5G and Beyond." Applied Sciences 11, no. 19 (2021): 9130.

Conference papers:

- Krzysztof Rusek, Paul Almasan, José Suárez-Varela, Piotr Chołda, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "Fast Traffic Engineering by Gradient Descent with Learned Differentiable Routing." In 2022 18th International Conference on Network and Service Management ***(CNSM)***, pp. 359-363.
- Carlos Güemes-Palau, Paul Almasan, Shihan Xiao, Xiangle Cheng, Xiang Shi, Pere Barlet-Ros, and Albert Cabellos-Aparicio. "Accelerating Deep Reinforcement Learning for Digital Twin Network Optimization with Evolutionary Strategies." In 2022 IEEE/IFIP Network Operations and Management Symposium ***(NOMS)***, pp. 1-5. IEEE, 2022.
- Arnau Badia-Sampera, José Suárez-Varela, Paul Almasan, Krzysztof Rusek, Pere Barlet-Ros, Albert Cabellos-Aparicio, "Towards more realistic network models based on Graph Neural Networks," in *Proceedings of the* ***ACM CoNEXT Student Workshop***, Orlando, FL, USA, Dec. 2019.

Demo papers:

- José Suárez-Varela, Sergi Carol-Bosch, Krzysztof Rusek, Paul Almasan, Marta Arias, Pere Barlet-Ros, and Albert Cabellos-Aparicio, "Challenging the generalization capabilities of Graph Neural Networks for network modeling," In *Proceedings of the **ACM SIGCOMM Conference Posters and Demos***, Beijing, China, Aug. 2019.

Technical reports:

- Paul Almasan, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangle Cheng, Albert Cabellos-Aparicio, and Pere Barlet-Ros. "Leveraging Spatial and Temporal Correlations for Network Traffic Compression." arXiv preprint arXiv:2301.08962 (2023).
- Paul Almasan, Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Diego Perino, Diego López, Antonio Agustin Pastor Perales et al. "Digital twin network: Opportunities and challenges." arXiv preprint arXiv:2201.01144 (2022).
- Paul Almasan, Jordi Paillisse, Alberto Rodriguez-Natal, Pere Barlet-Ros, Florin Coras, Vina Ermagan, Fabio Maino, and Albert Cabellos-Aparicio. "Securing the Control-plane Channel and Cache of Pull-based ID/LOC Protocols." arXiv preprint arXiv:1803.08568 (2018).

A.3 Other merits

- 3-month internship at Telefónica Research in Barcelona, Spain: working in collaboration with Dra. Andra Lutu and Dr. José Suárez-Varela to analyze real-world data to optimize mobile networks, from 2022-10-03 to 2023-01-02.
- 6-month research stay at the AGH University of Science and Technology in Krákow, Poland: working in collaboration with Prof. Krzysztof Rusek to investigate the application of Graph Neural Networks for network traffic compression, from 2022-04-02 to 2022-09-26.
- Doctoral fellowship from the Secretariat for Universities and Research of the Ministry of Business and Knowledge of the Government of Catalonia (ref. 2019 FI_B 00449), FI-AGAUR grant April 2019.

Bibliography

[1] P. Almasan, M. Ferriol-Galmes, J. Paillisse, J. Suarez-Varela, D. Perino, D. Lopez, A. A. P. Perales, P. Harvey, L. Ciavaglia, L. Wong, V. Ram, S. Xiao, X. Shi, X. Cheng, A. Cabellos-Aparicio, and P. Barlet-Ros, "Network digital twin: Context, enabling technologies and opportunities," *IEEE Communications Magazine*, pp. 1–13, 2022.

[2] H. Waldman, "The impending optical network capacity crunch," in *2018 SBFoton International Optics and Photonics Conference (SBFoton IOPC)*. Campinas, Brazil: IEEE, 2018, pp. 1–4.

[3] G. Wellbrock and T. J. Xia, "How will optical transport deal with future network traffic growth?" in *2014 The European Conference on Optical Communication (ECOC)*. Cannes, France: IEEE, 2014, pp. 1–3.

[4] A. Ellis, N. M. Suibhne, D. Saad, and D. Payne, "Communication networks beyond the capacity crunch," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2062, p. 20150191, 2016.

[5] Z. Zhang, Y. Xiao, Z. Ma, M. Xiao, Z. Ding, X. Lei, G. K. Karagiannidis, and P. Fan, "6g wireless networks: Vision, requirements, architecture, and key technologies," *IEEE Vehicular Technology Magazine*, vol. 14, no. 3, pp. 28–41, 2019.

[6] M. Giordani, M. Polese, M. Mezzavilla, S. Rangan, and M. Zorzi, "Toward 6g networks: Use cases and technologies," *IEEE Communications Magazine*, vol. 58, no. 3, pp. 55–61, 2020.

[7] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & information systems engineering*, vol. 6, no. 4, pp. 239–242, 2014.

[8] "Lte on the moon matters for networks on earth. https://www.nokia.com/networks/insights/network-on-the-moon/," accessed on: 2023-01-29. [Online]. Available: https://www.nokia.com/networks/insights/network-on-the-moon/

[9] H. Kim and N. Feamster, "Improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 114–119, 2013.

[10] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications surveys & tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[14] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[15] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[16] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire *et al.*, "Eta prediction with graph neural networks in google maps," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 3767–3776.

[17] S. Ravuri, K. Lenc, M. Willson, D. Kangin, R. Lam, P. Mirowski, M. Fitzsimons, M. Athanassiadou, S. Kashem, S. Madge *et al.*, "Skilful precipitation nowcasting using deep generative models of radar," *Nature*, vol. 597, no. 7878, pp. 672–677, 2021.

[18] A. S. Lundervold and A. Lundervold, "An overview of deep learning in medical imaging focusing on mri," *Zeitschrift für Medizinische Physik*, vol. 29, no. 2, pp. 102–127, 2019.

[19] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.

[20] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, 2003, pp. 3–10.

[21] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," *arXiv preprint arXiv:1710.11583*, 2017.

[22] P. Kalmbach, J. Zerwas, P. Babarczi, A. Blenk, W. Kellerer, and S. Schmid, "Empowering self-driving networks," in *Proceedings of the Afternoon Workshop on Self-Driving Networks*, 2018, pp. 8–14.

[23] H. X. Nguyen, R. Trestian, D. To, and M. Tatipamula, "Digital twin for 5g and beyond," *IEEE Communications Magazine*, vol. 59, no. 2, pp. 10–15, 2021.

[24] M. Ferriol-Galmés, J. Suárez-Varela, J. Paillissé, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Building a digital twin for network optimization using graph neural networks," *Computer Networks*, vol. 217, p. 109329, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128622003681

[25] Y. Wu, K. Zhang, and Y. Zhang, "Digital twin networks: A survey," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 13 789–13 804, 2021.

[26] C. Zhou, H. Yang, X. Duan, D. Lopez, A. Pastor, Q. Wu, M. Boucadair, and C. Jacquenet, "Digital Twin Network: Concepts and Reference Architecture," Internet Engineering Task Force, Internet-Draft draft-irtf-nmrg-network-digital-twin-arch-02, Oct. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-nmrg-network-digital-twin-arch/02/

[27] I. T. Union, "Recommendation ITU-T Y.3090: Digital twin network – Requirements and architecture," Feb. 2022, accessed on: 2023-01-31. [Online]. Available: https://handle.itu.int/11.1002/1000/14852

[28] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment." ICST, 5 2010.

[29] F. Xu, Y. Li, H. Wang, P. Zhang, and D. Jin, "Understanding mobile traffic patterns of large scale cellular towers in urban environment," *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 1147–1161, 2017.

[30] "The gzip home page. https://www.gzip.org/," accessed on: 2023-01-29. [Online]. Available: https://www.gzip.org/

[31] H. G. and R. M., "Pcap capture file format," https://tools.ietf.org/id/draft-gharris-opsawg-pcap-00.html, accessed on: 2023-01-29. [Online]. Available: https://tools.ietf.org/id/draft-gharris-opsawg-pcap-00.html

[32] P. Tune, M. Roughan, H. Haddadi, and O. Bonaventure, "Internet traffic matrices: A primer," *Recent Advances in Networking*, vol. 1, pp. 1–56, 2013.

[33] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 267–280. [Online]. Available: https://doi.org/10.1145/1879141.1879175

[34] K.-c. Lan and J. Heidemann, "A measurement study of correlations of internet flow characteristics," *Computer Networks*, vol. 50, no. 1, pp. 46–62, 2006.

[35] K. Gao, D. Li, L. Chen, J. Geng, F. Gui, Y. Cheng, and Y. Gu, "Predicting traffic demand matrix by considering inter-flow correlations," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 165–170.

[36] K. G. et al., "Incorporating intra-flow dependencies and inter-flow correlations for traffic matrix prediction," in *2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS)*, 2020, pp. 1–10.

[37] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com

[38] I. I. Cplex, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[39] L. Gong, X. Zhou, X. Liu, W. Zhao, W. Lu, and Z. Zhu, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, no. 8, pp. 836–847, 2013.

[40] M. Jadin, F. Aubry, P. Schaus, and O. Bonaventure, "Cg4sr: Near optimal traffic engineering for segment routing with column generation," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*. Paris, France: IEEE, 2019, pp. 1333–1341.

[41] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing ospf weights," in *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*, vol. 2. Tel Aviv, Israel: IEEE, 2000, pp. 519–528 vol.2.

[42] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A declarative and expressive approach to control forwarding paths in carrier-grade networks," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 15–28, Aug. 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787495

[43] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[44] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[45] M. T. Lucas, D. E. Wrege, B. J. Dempsey, and A. C. Weaver, "Statistical characterization of wide-area ip traffic," in *Proceedings of Sixth International Conference on Computer Communications and Networks*. IEEE, 1997, pp. 442–447.

[46] K. Thompson, G. J. Miller, and R. Wilder, "Wide-area internet traffic patterns and characteristics," *IEEE network*, vol. 11, no. 6, pp. 10–23, 1997.

[47] Z. Wang, Z. Li, G. Liu, Y. Chen, Q. Wu, and G. Cheng, "Examination of wan traffic characteristics in a large-scale data center network," in *Proceedings of the 21st ACM Internet Measurement Conference*, 2021, pp. 1–14.

[48] P. Shaw, "Using constraint programming and local search methods to solve vehicle routing problems," in *Principles and Practice of Constraint Programming — CP98*, M. Maher and J.-F. Puget, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 417–431.

[49] E. Nikolova, "High-performance heuristics for optimization in stochastic traffic engineering problems," in *Large-Scale Scientific Computing*, I. Lirkov, S. Margenov, and J. Waśniewski, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 352–360.

[50] E. Amaldi, A. Capone, L. G. Gianoli, and L. Mascetti, "A milp-based heuristic for energy-aware traffic engineering with shortest path routing," in *Network Optimization*, J. Pahl, T. Reiners, and S. Voß, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 464–477.

[51] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 197–210. [Online]. Available: https://doi.org/10.1145/3098822.3098843

[52] J. Perolat, B. D. Vylder, D. Hennes, E. Tarassov, F. Strub, V. de Boer, P. Muller, J. T. Connor, N. Burch, T. Anthony, S. McAleer, R. Elie, S. H. Cen, Z. Wang, A. Gruslys, A. Malysheva, M. Khan, S. Ozair, F. Timbers, T. Pohlen, T. Eccles, M. Rowland, M. Lanctot, J.-B. Lespiau, B. Piot, S. Omidshafiei, E. Lockhart, L. Sifre, N. Beauguerlange,

R. Munos, D. Silver, S. Singh, D. Hassabis, and K. Tuyls, "Mastering the game of stratego with model-free multiagent reinforcement learning," *Science*, vol. 378, no. 6623, pp. 990–996, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.add4679

[53] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz *et al.*, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.

[54] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, pp. 414–419, 2022.

[55] X. Chen, J. Guo, Z. Zhu, R. Proietti, A. Castro, and S. J. B. Yoo, "Deep-rmsa: A deep-reinforcement-learning routing, modulation and spectrum assignment agent for elastic optical networks," in *Proceedings of the Optical Fiber Communications Conference (OFC)*, 2018.

[56] J. Suárez-Varela, A. Mestres, J. Yu, L. Kuang, H. Feng, A. Cabellos-Aparicio, and P. Barlet-Ros, "Routing in optical transport networks with deep reinforcement learning," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 11, no. 11, pp. 547–558, 2019.

[57] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 191–205. [Online]. Available: https://doi.org/10.1145/3230543.3230551

[58] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XVI. New York, NY, USA: Association for Computing Machinery, 2017, p. 185–191. [Online]. Available: https://doi.org/10.1145/3152434.3152441

[59] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[60] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[61] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[62] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[63] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[64] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[65] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[66] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated graph sequence neural networks," *arXiv preprint arXiv:1511.05493*, 2015.

[67] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[68] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the International Conference on Machine Learning (ICML) - Volume 70*, 2017, pp. 1263–1272.

[69] P. W. Battaglia, R. Pascanu, M. Lai, D. J. Rezende *et al.*, "Interaction networks for learning about objects, relations and physics," in *Proceedings of Advances in neural information processing systems (NIPS)*, 2016, pp. 4502–4510.

[70] Z. Pan, Y. Liang, W. Wang, Y. Yu, Y. Zheng, and J. Zhang, "Urban traffic prediction from spatio-temporal data using deep meta learning," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 1720–1730.

[71] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[72] L. Bai, L. Yao, C. Li, X. Wang, and C. Wang, "Adaptive graph convolutional recurrent network for traffic forecasting," *Advances in neural information processing systems*, vol. 33, pp. 17 804–17 815, 2020.

[73] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, p. 529–533, 2015.

[74] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," *IEEE Network*, vol. 32, no. 2, pp. 92–99, 2017.

[75] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. Honolulu, HI, USA: IEEE, 2018, pp. 1871–1879.

[76] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 conference of the ACM special interest group on data communication*, 2018, pp. 191–205.

[77] A. Mestres, E. Alarcón, Y. Ji, and A. Cabellos-Aparicio, "Understanding the modeling of computer network delays using neural networks," in *Proceedings of the ACM SIGCOMM Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (Big-DAMA)*, 2018, pp. 46–52.

[78] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner *et al.*, "Relational inductive biases, deep learning, and graph networks," *arXiv preprint arXiv:1806.01261*, 2018.

[79] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet: Leveraging graph neural networks for network modeling and optimization in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2260–2270, 2020.

[80] P. Almasan, "Code of DRL+GNN architecture in OTN," 11 2021, accessed on: 2023-01-31. [Online]. Available: https://github.com/knowledgedefinednetworking/DRL-GNN

[81] J. Kuri, N. Puech, and M. Gagnaire, "Diverse routing of scheduled lightpath demands in an optical transport network," in *Proceedings of the IEEE International Workshop on Design of Reliable Communication Networks (DRCN)*, 2003, pp. 69–76.

[82] "Itu-t recommendation g.709/y.1331: Interface for the optical transport network," 2019, https://www.itu.int/rec/T-REC-G.709/.

[83] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[84] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[85] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016, pp. 265–283.

[86] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[87] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of the International Conference on Computational Statistics (COMPSTAT)*, 2010, pp. 177–186.

[88] X. Hei, J. Zhang, B. Bensaou, and C.-C. Cheung, "Wavelength converter placement in least-load-routing-based optical networks using genetic algorithms," *Journal of Optical Networking*, vol. 3, no. 5, pp. 363–378, 2004.

[89] F. Barreto, E. C. Wille, and L. Nacamura Jr, "Fast emergency paths schema to overcome transient link failures in ospf routing," *arXiv preprint arXiv:1204.2465*, 2012.

[90] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving sub-second igp convergence in large ip networks," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, pp. 35–44, 2005.

[91] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[92] D. A. S. Aric A. Hagberg and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[93] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[94] Z. Yang, Y. Cui, B. Li, Y. Liu, and Y. Xu, "Software-defined wide area network (sd-wan): Architecture, advances and opportunities," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–9.

[95] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A roadmap for traffic engineering in sdn-openflow networks," *Computer Networks*, vol. 71, pp. 1–30, 2014.

[96] B. Fortz and M. Thorup, "Optimizing ospf/is-is weights in a changing world," *IEEE journal on selected areas in communications*, vol. 20, no. 4, pp. 756–767, 2002.

[97] R. Bhatia, F. Hao, M. Kodialam, and T. Lakshman, "Optimized network traffic engineering using segment routing," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. Hong Kong, China: IEEE, 2015, pp. 657–665.

[98] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, "The segment routing architecture," in *2015 IEEE Global Communications Conference (GLOBECOM)*. San Diego, CA, USA: IEEE, 2015, pp. 1–6.

[99] Q. Cappart, T. Moisan, L.-M. Rousseau, I. Prémont-Schwarz, and A. Cire, "Combining reinforcement learning and constraint programming for combinatorial optimization," *arXiv preprint arXiv:2006.01610*, 2020.

[100] J. Zhang, M. Ye, Z. Guo, C.-Y. Yen, and H. J. Chao, "Cfr-rl: Traffic engineering with reinforcement learning in sdn," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2249–2259, 2020.

[101] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 561–575. [Online]. Available: https://doi.org/10.1145/3230543.3230544

[102] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 334–350. [Online]. Available: https://doi.org/10.1145/3341302.3342076

[103] Q. Huang, X. Jin, P. P. C. Lee, R. Li, L. Tang, Y.-C. Chen, and G. Zhang, "Sketchvisor: Robust network measurement for software packet processing," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 113–126. [Online]. Available: https://doi.org/10.1145/3098822.3098831

[104] Z. Ahmed, N. Le Roux, M. Norouzi, and D. Schuurmans, "Understanding the impact of entropy on policy optimization," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 151–160. [Online]. Available: http://proceedings.mlr.press/v97/ahmed19a.html

[105] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[106] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science and Engg.*, vol. 13, no. 2, p. 31–39, Mar. 2011. [Online]. Available: https://doi.org/10.1109/MCSE.2010.118

[107] P. Almasan, "Enero code and datasets for routing optimization in IP networks," 7 2022, accessed on: 2023-01-31. [Online]. Available: https://github.com/BNN-UPC/ENERO

[108] S. Gay, P. Schaus, and S. Vissicchio, "Repetita: Repeatable experiments for performance evaluation of traffic-engineering algorithms," *arXiv preprint arXiv:1710.08665*, 2017.

[109] U. Brandes, "On variants of shortest-path betweenness centrality and their generic computation," *Social networks*, vol. 30, no. 2, pp. 136–145, 2008.

[110] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 123–137, aug 2015. [Online]. Available: https://doi.org/10.1145/2829988.2787472

[111] W. Fang and L. Peterson, "Inter-as traffic patterns and their implications," in *Seamless Interconnection for Universal Services. Global Telecommunications Conference. GLOBECOM'99.(Cat. No. 99CH37042)*, vol. 3. IEEE, 1999, pp. 1859–1868.

[112] T. Mori, R. Kawahara, S. Naito, and S. Goto, "On the characteristics of internet traffic variability: spikes and elephants," *IEICE TRANSACTIONS on Information and Systems*, vol. 87, no. 12, pp. 2644–2653, 2004.

[113] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, no. 6, p. 520–540, jun 1987. [Online]. Available: https://doi.org/10.1145/214762.214771

[114] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "SNDlib 1.0—survivable network design library," *Networks: An International Journal*, vol. 55, no. 3, pp. 276–286, 2010.

[115] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[116] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," in *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 219–230. [Online]. Available: https://doi.org/10.1145/1015467.1015492

[117] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI'18. AAAI Press, 2018, p. 3634–3640.

[118] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[119] Nayuki, "https://github.com/nayuki/reference-arithmetic-coding," accessed on: 2023-01-29. [Online]. Available: https://github.com/nayuki/Reference-arithmetic-coding

[120] S. Seabold and J. Perktold, "Statsmodels: Econometric and statistical modeling with python," in *Proceedings of the 9th Python in Science Conference*, vol. 57. Austin, TX, 2010, pp. 10–25 080.

[121] D. A. Dickey and W. A. Fuller, "Distribution of the estimators for autoregressive time series with a unit root," *Journal of the American statistical association*, vol. 74, no. 366a, pp. 427–431, 1979.

[122] L. Gong, X. Zhou, W. Lu, and Z. Zhu, "A two-population based evolutionary approach for optimizing routing, modulation and spectrum assignments (rmsa) in o-ofdm networks," *IEEE Communications letters*, vol. 16, no. 9, pp. 1520–1523, 2012.

[123] M. Klinkowski, M. Ruiz, L. Velasco, D. Careglio, V. Lopez, and J. Comellas, "Elastic spectrum allocation for time-varying traffic in flexgrid optical networks," *IEEE journal on selected areas in communications*, vol. 31, no. 1, pp. 26–38, 2012.

[124] S. Gay, R. Hartert, and S. Vissicchio, "Expect the unexpected: Sub-second optimization for segment routing," in *IEEE Conference on Computer Communications (INFOCOM)*. Atlanta, GA, USA: IEEE, 2017, pp. 1–9.

[125] Y. Wang, X. Cao, and Y. Pan, "A study of the routing and spectrum allocation in spectrum-sliced elastic optical path networks," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2011, pp. 1503–1511.

[126] K. Christodoulopoulos, I. Tomkos, and E. A. Varvarigos, "Elastic bandwidth allocation in flexible ofdm-based optical networks," *Journal of Lightwave Technology*, vol. 29, no. 9, pp. 1354–1366, 2011.

[127] P. Sun, Z. Guo, J. Lan, J. Li, Y. Hu, and T. Baker, "Scaledrl: a scalable deep reinforcement learning approach for traffic engineering in sdn with pinning control," *Computer Networks*, vol. 190, p. 107891, 2021.

[128] S. Troia, F. Sapienza, L. Varé, and G. Maier, "On deep reinforcement learning for traffic engineering in sd-wan," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 7, pp. 2198–2212, 2020.

[129] F. Geyer and G. Carle, "Learning and generating distributed routing protocols using graph-based deep learning," in *Proceedings of the ACM SIGCOMM Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (Big-DAMA)*, 2018, pp. 40–45.

[130] H. Zhu, V. Gupta, S. S. Ahuja, Y. Tian, Y. Zhang, and X. Jin, "Network planning with deep reinforcement learning," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 258–271.

[131] G. Bernárdez, J. Suárez-Varela, A. López, B. Wu, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Is machine learning ready for traffic engineering optimization?" in *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE, 2021, pp. 1–11.

[132] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of ACM SIGCOMM*, 2019, pp. 270–288.

[133] W. Aiello, A. Gilbert, B. Rexroad, and V. Sekar, "Sparse approximations for high fidelity compression of network traffic data," in *Proceedings of the 5th ACM SIGCOMM conference on Internet measurement*, 2005, pp. 22–22.

[134] A. Beirami, M. Sardari, and F. Fekri, "Packet-level network compression: Realization and scaling of the network-wide benefits," *IEEE/ACM Transactions on Networking*, vol. 24, no. 3, pp. 1588–1604, 2015.

[135] R. Mehboob, S. A. Khan, and Z. Ahmed, "High speed lossless data compression architecture," in *2006 IEEE International Multitopic Conference*. IEEE, 2006, pp. 84–88.

[136] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[137] F. Fusco, M. P. Stoecklin, and M. Vlachos, "Net-fli: on-the-fly compression, archiving and indexing of streaming network traffic," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 1382–1393, 2010.

[138] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese *et al.*, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[139] R. Vinayakumar, K. Soman, and P. Poornachandran, "Applying deep learning approaches for network traffic prediction," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. IEEE, 2017, pp. 2353–2358.

[140] N. Ramakrishnan and T. Soni, "Network traffic prediction using recurrent neural networks," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2018, pp. 187–193.

[141] H. Yang, X. Li, W. Qiang, Y. Zhao, W. Zhang, and C. Tang, "A network traffic forecasting method based on sa optimized arima–bp neural network," *Computer Networks*, vol. 193, p. 108102, 2021.

[142] D. Andreoletti, S. Troia, F. Musumeci, S. Giordano, G. Maier, and M. Tornatore, "Network traffic prediction based on diffusion convolutional recurrent neural networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 246–251.

[143] M. Li, Y. Wang, Z. Wang, and H. Zheng, "A deep learning method based on an attention mechanism for wireless network traffic prediction," *Ad Hoc Networks*, vol. 107, p. 102258, 2020.