UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Multi-agent graph learning-based optimization and its applications to computer networks

Guillermo Bernárdez Gil

Multi-Agent Graph Learning-based Optimization
and its Applications to Computer Networks

Thesis for the Doctoral Degree in Computer Architecture

by

Guillermo Bernárdez Gil

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

November 2023

Departament d'Arquitectura de Computadors (DAC)
Universitat Politècnica de Catalunya (UPC)

Abstract

In the wake of a digital revolution, contemporary society finds itself entrenched in an era where network applications' demands surpass the capabilities of conventional network management solutions. This dissertation navigates through the intricacies of modern networked environments, where traditional management approaches are falling short due to emerging applications like augmented and virtual reality, holographic telepresence, and vehicular networks, demanding ultra-low latency and robust adaptability. These evolving networks form the backbone of modern society, sustaining numerous vital services but posing elevated complexity and operational hurdles for Internet Service Providers (ISPs) and network operators.

Amidst this complexity, the need for innovative solutions to optimize and manage today's networks is more pronounced than ever. A central proposition of this dissertation is the MAGNNETO framework, a groundbreaking Machine Learning (ML) based initiative that stands for Multi-Agent Graph Neural Network Optimization. This framework is at the heart of the endeavour to facilitate distributed optimization in networked scenarios. By integrating a Graph Neural Network (GNN) architecture into a Multi-Agent Reinforcement Learning (MARL) setting, it instigates a fully distributed optimization process and capitalizes on the inherent distributed nature of networked environments, hence potentially addressing scalability issues and facilitating real-time applications. This initiative is adaptable, offering versatility in addressing various use cases and showcasing robustness to meet the challenging requisites of real-world applications.

A substantial contribution of this work is the successful implementation of MAGNNETO across different relevant networked cases, prominently focusing on two highly impactful scenarios within the computer network field. Initially, it re-examines the pivotal issue of Traffic Engineering (TE) optimization in ISP networks. With the goal of curtailing network congestion, MAGNNETO-TE is introduced, a variant of the framework specifically devised to minimize maximum link utilization in these networks. Remarkably, this adaptation heralds a paradigm shift by equalling the performance of traditional state-of-the-art TE optimizers but at a fraction of the execution cost.

Moreover, the research explores the complex sphere of Congestion Control (CC) in Datacenter Networks (DCN), another critical service in our current digital world that is characterized by dynamic traffic patterns and stringent low-latency prerequisites. Here, MAGNNETO-CC emerges as a potent solution, offering an offline, distributed strategy that harmonizes with widely deployed CC protocols, surpassing other state-of-the-art ML-based CC methodologies and prevailing static CC configurations in performance.

Looking ahead, the dissertation also delineates potential avenues to enhance MAGNNETO, particularly addressing challenges tied to current GNN architectures (e.g. over-smoothing and over-squashing). It envisions integrating Topological Deep Learning (TDL) techniques to foster a novel, promising approach to distributed optimization that has the potential to exploit arbitrary multi-element correlations, going beyond the traditional graph domain. By addressing the urgent need for efficient network traffic storage on networks with multiple vantage points, the proposed topological-inspired methodology reveals itself as a robust ML-based baseline for lossy data compression.

In summation, this dissertation embarks on a pioneering journey to confront the elemental challenges of optimizing networked, graph-based systems. It unfurls the innovative MAGNNETO solution as a beacon of versatility and adaptability, displays its multifaceted applications, and heralds promising directions for future research, aiming to redefine the landscape of distributed network optimization and management in this digitally transformative era.

Acknowledgments

First of all, I would like to address my advisors, Prof. Albert Cabellos and Prof. Pere Barlet. Congratulations for your amazing work, I couldn't be more grateful for your supervision and mentoring, for your passionate implication and guidance, for the knowledge and advice you are always eager to offer. You have helped me grow in many aspects during this exciting journey, and I will always be indebted to you for it. Thank you!

I would also like to extend my profound gratitude to Prof. Pietro Liò for welcoming and hosting me at the University of Cambridge during my six-month stay. Your kind hospitality afforded me the opportunity to connect with remarkable people and has significantly influenced my future research trajectory. My profound thanks to Prof. Eduard Alarcón as well, who made this entire experience possible in the first place.

What a wonderful time in Cambridge...! Thank you to all the students and visitors that I had the privilege to meet and collaborate with during that period. Among them, my special gratitude to Olga, Francesco (×2), Lisa, Federico, and Maria Sofia for all the great memories we share. And I cannot end this paragraph without mentioning you, Lev: not only you have become my closest collaborator, but also one of my best friends. Thank you, man!

Heartfelt appreciation also goes to every member, past and present, of the Barcelona Neural Networking Center and the Computer Architecture Department. Special recognition is owed to Jose Suárez Varela, Paul Almasan, Jordi Paillissé Vilanova, Sergi Abadal, Albert López, Miquel Ferriol Galmés, Carlos Güemes Palau, Berta Serracanta, Hamid Latif and Ángela López. I will never forget all the moments we have shared, specially our team building meetings (you have my axe!).

My friends, your unwavering love and encouragement have been my anchor during the challenges of this PhD journey. I am infinitely blessed to have you in my life.

I would also like to acknowledge the Spanish I+D+i project TRAINER-A (ref. PID2020-118011GB-C21), funded by MCIN/ AEI/10.13039/501100011033, and the Secretariat for Universities and Research of the Ministry of Business and Knowledge of the Government of Catalonia, in collaboration with the European Social Fund. Their financial support made both this PhD and my research stay at the University of Cambridge possible.

Finalmente, quiero dedicar estas últimas palabras directamente a mi familia, sobre todo a mis padres y abuelos: gracias por estar siempre a mi lado, gracias por apoyarme incondicionalmente en todos mis proyectos. De nuevo, esto va por y para vosotros.

Y especialmente a ti, mama: te quiero.

Contents

List of Acronyms

CC Congestion Control.

CDF Cumulative Distribution Function.

CNN Convolutional Neural Network.

CP Constraint Programming.

DCN Datacenter Network.

DCQCN Datacenter Quantized Congestion Notification.

DCTCP Datacenter Transmission Control Protocol.

Dec-POMDP Decentralized Partially Observable Markov Decision Process.

DQN Deep Q-learning Network.

ECMP Equal Cost Multi-Path.

ECN Explicit Congestion Notification.

FCT Flow Completion Time.

GAE Generalized Advantage Estimate.

GNN Graph Neural Network.

IGP Interior Gateaway Protocol.

ILP Integer Linear Programming.

ISP Internet Service Provider.

LS Local Search.

LSA Link-State Advertisement.

MARL Multi-Agent Reinforcement Learning.

MDP Markov Decision Process.

ML Machine Learning.

MLP Multi-Layer Perceptron.

MP Message Passing.

MPNN Message Passing Neural Network.

NM Numerical Methods.

NN Neural Network.

OSPF Open Shortest Path First.

PG Policy Gradient.

PPO Proximal Policy Optimization.

RDMA Remote Direct Memory Access.

RL Reinforcement Learning.

SGD Stochastic Gradient Descent.

SR Segment Routing.

TCP/IP Transmission Control Protocol/Internet Protocol.

TDL Topological Deep Learning.

TE Traffic Engineering.

TM Traffic Matrix.

TNN Topological Neural Network.

TRPO Trust Region Policy Optimization.

List of Figures

Page

List of Tables

Chapter 1

Introduction

The profound digital transformation sweeping across society and industry has ushered in a new era of network applications with intricate requirements that surpass the capabilities of traditional network management solutions. These emerging applications, spanning diverse domains from augmented and virtual realities to holographic telepresence and vehicular networks, demand ultra-low deterministic latency, real-time adaptability, and robustness in the face of dynamic, heterogeneous networks. Simultaneously, the explosive growth in the number of connected devices has rendered modern networks more dynamic and diverse than ever before, leading to increased complexity and operational costs in managing them [1–3].

Communication networks have evolved to become the backbone of contemporary society, underpinning a wide range of critical services: government services, education, healthcare, banking and e-commerce, social interactions, etc. However, this transformation has come at the cost of complexity and heightened operational challenges for Internet Service Provider (ISP) networks tasked with maintaining network performance and service-level agreements for a multitude of users and applications.

The trajectory for the coming years points toward further escalation in the number of connected devices and the volume of network traffic. Initiatives like the deployment of 5G networks in numerous countries and the exploration of 6G networks signify a continuing trend. These networks are expected to not only connect people but also encompass sensors, vehicles, robots, and computational resources [4, 5]. Industrial projections like Industry 4.0 are set to benefit from the interconnection of factories and machinery [6], while ambitious projects extend terrestrial communication networks beyond Earth's boundaries. Examples include SpaceX's Starlink project, which leverages satellite connectivity to enhance internet access in remote areas, and the collaboration between NOKIA and NASA to establish the first LTE network on the moon [7].

With the emergence of these diverse applications, each accompanied by stringent network prerequisites such as ultra-low latency, the task of administering modern communication networks has become substantially more complex. Consequently, network operators find themselves at a crossroads, seeking innovative strategies to manage these burgeoning heterogeneous networks proficiently.

In parallel, the field of Machine Learning (ML) saw a series of breakthroughs in the last decade that are also leading into a new and transformative era. Innovations like the Convolutional Neural Network (CNN) architecture [8], AlphaGo [9], BERT [10], and the more recent AlphaFold [11] showcased the remarkable capabilities of ML models, outperforming conventional solutions in various domains. These advancements spurred interest in using ML to address real-world challenges, ranging from estimating arrival times in navigation apps to improving medical imaging [12].

This dissertation aims to explore the intricate landscape of modern networked environments, where traditional management approaches fall short, and to investigate innovative solutions based on recent advances in ML to address the challenges of efficiently managing complex, heterogeneous and inherently distributed networks.

1.1 Motivation and Objectives

In general, the optimization of distributed systems is a fundamental challenge that underpins the efficient operation of a wide range of scenarios: from all sorts of computer networks to social networks, traffic lights in smart cities, autonomous vehicles, power grids, etc. This optimization problem involves finding optimal configurations and strategies to improve the performance of these complex interconnected systems. While numerous methods and techniques have been proposed to address this challenge [13], the focus has often been on specific use cases or centralized approaches that suffer from computational inefficiency and scalability limitations [14].

In fact, the optimization of networked scenarios has been traditionally approached as a combinatorial problem [15–19], employing classical methodologies such as Integer Linear Programming (ILP), Local Search (LS), Constraint Programming (CP), or Numerical Methods (NM). While these centralized methods offer theoretical guarantees, they are computationally expensive and struggle to scale with the size of the network graph. Consequently, they often yield suboptimal solutions unless significant computational resources are expended. Moreover, their lack of scalability precludes their use in real-time applications, where optimization needs to adapt rapidly to changes in the network [20].

In contrast to centralized approaches, the design of decentralized architectures capitalizes on the inherent distributed nature of such networked environments. This approach holds the promise of mitigating scalability challenges by parallelizing optimization processes across computationally-enabled elements of the network. ML-based solutions, and in particular Multi-Agent Reinforcement Learning (MARL) methodologies [21], have attracted considerable attention in recent years as they perfectly fit into these distributed design principles. However, many existing methods in this domain employ a standard Neural Network

(NN) architecture to model policies, often leading to isolated agents with limited cooperation or relying on ad-hoc communication mechanisms tailored to specific use cases [22–25].

The main motivation of this dissertation is to explore and address the general problem of optimizing distributed systems leveraging the promising MARL methodology together with a Graph Neural Network (GNN) [26], which might unveil the potential of exploiting graph-based inductive biases of networked environments and further benefit from the distributed nature of such networks. Prior to our initial research, there were few MARL frameworks that embedded GNNs to model distributed scenarios, and the found references were limited to AI-focused papers with no realistic applications [27, 28].

Hence, we aimed to design a novel framework combining MARL and GNNs able to optimize networked systems in a distributed way by deploying a set of agents across the network equipped with topology-awareness and generalization capabilities. Moreover, the idea is that the framework is not confined to a specific application domain, and to do so we should carefully design general, use case-agnostic cooperation mechanisms among agents that can easily adapt to different distributed systems and which are robust enough to satisfy the challenging requirements of real-world applications (low latency, feasible link overloads, compatible with legacy hardware, etc.) [20, 29].

Our research has been specially motivated by the need of innovative solutions for the optimization and management of today's computer networks, which as we exposed above posit very hard challenges for network operators due to the observed exponential growth and importance in our society. Thus, it is our objective to show that such a framework can be adapted to operate different real-world computer network scenarios and can be successfully compared against the state-of-the-art on those use cases. In addition to that, we also have the long-term goal to show that our framework can also be considered in other distributed systems beyond computer-based infrastructures.

Finally, the dissertation also delves into potential future directions for enhancing this framework, particularly addressing challenges related to current GNN models. To this end, the objective is to explore the integration of very recent Topological Deep Learning (TDL) [30, 31] techniques, which have shown promise in handling complex relational data and long-range interactions, in order for our framework to go beyond the graph-based representations and local neighborhoods that GNNs rely on.

1.2 Contributions

The first and main contribution of this dissertation is our proposed MAGNNETO framework for distributed optimization in networked scenarios, which stands for Multi-Agent Graph Neural Network Optimization. By embedding a GNN architecture into a MARL setting, MAGNNETO is designed to optimize networked environments in a fully distributed manner by naturally parallellizing the optimization process among the computationally-enabled nodes of a network –where agents are deployed and operate. It can be easily adapted to different use cases by properly adjusting the RL environment (state and action spaces, reward function), and due to its modular GNN-based modeling of the system it can easily generalize over scenarios not previously seen in training and be fully compatible with the hardware of today's networked infrastructures.

The second substantial contribution is precisely the successful application of MAGNNETO to different networked use cases. This dissertation addresses two impactful and challenging optimization scenarios within computer networks, ultimately showing that MAGNNETO provides with relevant contributions and improvements in both active research fields.

First, we revisit the fundamental problem of Traffic Engineering (TE) optimization in ISP networks, where the goal is to minimize network congestion. Although many proposals had tackled this challenge from various angles, ML-based solutions had yet to surpass traditional optimization algorithms. In this context, we introduce MAGNNETO-TE, an adaptation of our framework tailored for minimizing the maximum link utilization of such networks. We show for the first time that a ML-based approach can obtain similar performance than classical state-of-the-art TE optimizers while significantly reducing the execution cost of those traditional, centralized solutions.

Next, we also consider the challenging optimization scenario of Congestion Control (CC) in a Datacenter Network (DCN), a domain marked by dynamic traffic patterns and low-latency requirements. While ML-based solutions have made strides, they often require network stack reimplementations or online training, limiting their applicability. On the other hand, our MAGNNETO-CC adaptation offers an offline, distributed solution that is compatible with widely deployed CC protocols. In our evaluation, we show that MAGNNETO-CC outperforms state-of-the-art ML-based CC methodologies, as well as static CC configuration baselines that are typically used in current DCNs.

Last, but not least, the dissertation also delves into potential future directions for enhancing MAGNNETO, particularly addressing challenges related to GNN over-smoothing and over-squashing issues. We explore the integration of TDL techniques, which have shown promise in handling complex relational data and long-range interactions, offering a novel approach to distributed optimization in networked scenarios. In this last contribution, we show that our proposed TDL-based methodologies can potentially define a strong baseline for lossy data compression by exploiting multi-element correlations and going beyond the graph domain. The obtained results in network traffic compression on networks with multiple vantage points reveal that our models outperform other ML-based architectures (including different GNNs) for this task.

1.3 Outline of the Thesis

In summary, this dissertation embarks on a journey to tackle the fundamental problem of optimizing networked, graph-based environments, first presenting MAGNNETO as a novel and versatile solution, then showcasing its different applications, and finally exposing the conclusions as well as avenues for future research and enhancement. These 3 steps define the general outline of the Thesis, which we detail as follows:[1]

Proposed Method

Section 2: Background

In this section we provide the technical background of our proposed architecture, describing the fundamentals of the MARL and GNN technologies it is based on.

Section 3: Multi-Agent Graph Neural Network Optimization

This section introduces the MAGNNETO framework, our general multi-agent optimization architecture for distributed systems. After reviewing the related literature, the pipeline of MAGNNETO is described in detail, and the section concludes with a summary of its main features and contributions.
Related Outcomes: Publications [32–34] & Patents [35, 36].

Applications

Section 4: MAGNNETO-TE: Traffic Engineering in ISP Networks

This section is devoted to describe our first application of MAGNNETO to a real-world scenario: Traffic Engineering in ISP networks. The section motivates and outlines this network scenario, and then introduces and evaluates MAGNNETO-TE, the adaptation of the framework to this specific task.
Related Outcomes: Publications [32, 33] & Patent [35].

Section 5: MAGNNETO-CC: Congestion Control in Datacenters

This section presents our second application of MAGNNETO, now to Congestion Control optimization in DCNs. We deep into this particular network scenario, present an adaptation of the framework –MAGNNETO-CC– that addresses the challenges it poses, and perform an extensive evaluation of the model.
Related Outcomes: Publication [34] & Patent [36].

[1] More details about the related outcomes of each chapter –both publications and submitted patents– can be found in Appendix A.

Future Directions & Conclusions

Section 6: Beyond the Graph Domain: Topological Network Traffic Compression

This section explores the use of TDL methods to extend MAGNNETO's architecture beyond graph-based representations, and to do so we consider the relevant use case of traffic data compression on networks with multiple vantage points.
Related Outcomes: Publications [37, 38].

Section 7: Conclusions and Future Work

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

Proposed Method

Chapter 2

Background

The distributed architecture proposed in this thesis combines two Machine Learning (ML) mechanisms. First, we use a Graph Neural Network (GNN) to model networked scenarios. GNNs are neural network architectures specifically designed to generalize over graph-structured data [39], and thus, are well suited to operate successfully in other network scenarios including topologies and configurations unseen during training. Moreover, they offer near real-time operation below the scale of milliseconds.

Second, we design a Multi-Agent Reinforcement Learning (MARL) setting -which highly relies on the more fundamental theory of single-agent Reinforcement Learning (RL)- to build a distributed system in which each node or link becomes an agent that, having access only to local information, learns how to efficiently modify its internal attributes in order to achieve a global optimization goal.

This chapter provides a technical background of these ML-based methodologies.

2.1 Graph Neural Networks

GNNs are a novel family of neural networks designed to operate over graphs. They were introduced in [26] and numerous variants have been developed since [40–44]. In their basic form, they consist in associating some initial states to the different elements of an input graph, and combine them considering how these elements are connected in the graph. An iterative message-passing 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 Network (MPNN) [45,46] are a well-known type of GNNs that apply an iterative message-passing algorithm to propagate information between the nodes of the graph. For clarity, we omit edge- and graph-level features and describe the general

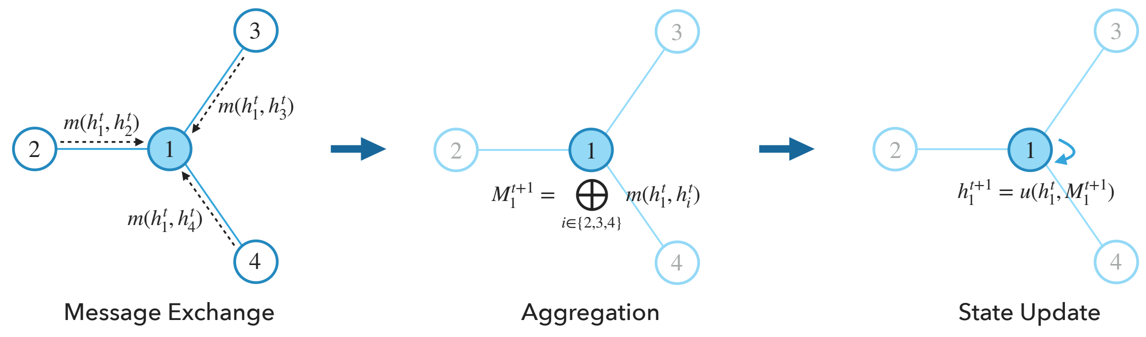


Figure 2.1: Message passing overview from the perspective of a single node.

execution on a node level basis. Formally, let a graph be a tuple of nodes and edges, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and denote by $h_k^t \in \mathbb{R}^d$ the hidden state of a node k at message passing iteration t –whose initializations h_k^0 typically leverage the initial node features $x_k \in \mathbb{R}^f$ if available.

In a message-passing step (see Figure 2.1), each node k receives messages from all the nodes in its one-hop neighborhood $N(k) = \{v \in \mathcal{V} \mid (k, v) \in \mathcal{E}\}$. Messages are generated by applying a message function $m(\cdot)$ to the hidden states of node pairs in the graph. Then, they are combined by a permutation invariant aggregation function $\oplus$ (Equation 2.1.1)– for instance, an element-wise sum, mean or min/max. Finally, an update function $u(\cdot)$ is used to compute a new hidden state for every node (Equation 2.1.2):

$$M_k^{t+1} = \bigoplus_{i\in N(k)} m(h_k^t, h_i^t), \tag{2.1.1}$$

$$h_k^{t+1} = u(h_k^t, M_k^{t+1}), \tag{2.1.2}$$

where $m(\cdot)$ and $u(\cdot)$ are differentiable functions, and consequently may be learned by neural networks. After a certain number of iterations T, the final hidden node representations $\{h_k^T\}_{k\in\mathcal{V}}$ are used by a readout function $r(\cdot)$ to produce an output for the given task. This function can also be implemented by a neural network and is typically tasked to predict either individual properties of nodes (e.g., its class) or global properties at the graph level.

Notably, the same message, aggregation and update functions –as well as the read-out if it predicts a node property– are applied to each node; this feature, together with permutation invariant aggregators, unveil the potential of MPNNs to exploit graph-based inductive biases, enabling unprecedented scalability/generalization capabilities in the graph domain. In essence, the motivation behind GNNs is to provide a learning framework that can naturally handle the complexities and irregularities of graph data, encompassing both its topological structure and feature information. As graph-structured data is ubiquitous in the real world, the relevance and importance of GNNs have grown exponentially in a wide range of domains of science and technology [47].

2.2 Reinforcement Learning

The concept of RL [48] emerged with the objective of designing agents able to deal with sequential decision problems by themselves, in the sense that they are not told which actions to take; instead, they must learn good long-term policies by optimizing a cumulative future reward signal.

In the standard RL setting [49], an agent interacts with the environment in the following way: at each step t, the agent selects an action a_t based on its current state s_t, to which the environment responds with a reward r_t and then moves to the next state s_{t+1}. This interaction is modeled as a time-homogeneous Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, r, P, \gamma)$, where

- $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively. It is assumed that both are finite, with $n := |\mathcal{S}|$;
- P is the transition kernel, $s_{t+1} \sim P(\cdot|s_t, a_t)$; the Markov assumption states that $P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ...) = P(s_{t+1}|s_t, a_t)$;
- r_t represents the immediate reward given by the environment after taking action a_t being in state s_t. These rewards are considered to be sampled from a reward distribution $R(s, a)$, i.e. $r_t \sim R(s_t, a_t)$;
- γ is the discount factor.

Another relevant concept is that of the return, usually defined as the discounted cumulative rewards along a certain agent trajectory $\{(s_t, a_t, r_t)\}_{t=0}^{T}$, i.e.

$$G_t = \sum_{t=0}^{T} \gamma^t R(s_t, a_t) \tag{2.2.1}$$

Finally, the behaviour of the agent is described by a policy $\pi : \mathcal{S} \to \mathcal{A}$, which maps each state to a probability distribution over the action space. Hence, the goal of a RL agent is to find the optimal policy in the sense that, given any considered state $s \in \mathcal{S}$, it always select an action that maximizes the expected return.

There are two main model-free approaches to this end [48]:

- Action-value methods, typically referred to as Q-learning; the policy π is indirectly defined from the learned estimates of the action value function

$$Q^{\pi}(s, a) \;=\; \mathop{\mathbb{E}}_{\pi}\left[G_t | s_0 = s, a_0 = a\right].$$

- Policy Gradient (PG) methods, which directly attempt to learn a parameterized policy representation π_θ. The Actor-Critic family of PG algorithms also involves learning a function approximator $V_\phi(s)$ of the state value function $V^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}[G_t | s_t = s]$. In this case, actions are exclusively selected from function π_θ, which estimates the policy (i.e., the actor), but the training of such policy is guided by the estimated value function $V_\phi(s)$, which assesses the consequences of the actions taken (i.e., the critic).

We provide more details about each of these RL branches below.

2.2.1 Q-learning Methods

Q-learning methods consider the state-action value function (a.k.a. q-value function), defined as the expected discounted return from a state-action pair by following a certain policy π, i.e.

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s, a_0 = a \right] \tag{2.2.2}$$

As shown in [48], looking for a policy with an optimal state-value function for every state-action pair is equivalent to finding the optimal policy. This is precisely the basis of all RL methods covered by the names of Q-learning and Q-networks.

In fact, Q-learning is as well the name of the first algorithm that sought for the optimal state-value function to solve the RL problem setting [50]. In particular, Q-learning is a tabular method -i.e. constructs a table with all possible combinations of state-action pairs- that obtains the optimal state-action value function by iteratively updating

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big(r(s_t, a_t) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \Big), \tag{2.2.3}$$

where $\alpha \in (0, 1]$ is the learning rate and $\gamma \in [0, 1]$ the discount factor. However, the use of tabular representations is unfeasible when dealing with large state spaces, which is the case of most RL problems.

Deep Q-learning Network (DQN) [51] overcomes this limitation by designing a Neural Network (NN) as a function approximator for the q-value function. This enables to rely on the generalization capabilities of NNs to estimate the q-values of states and actions unseen in advance. For this reason, a NN 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 trajectory tuples of the form $\{s_t, a_t, r_t, s_{t+1}\}$). While training the neural network, the temporal correlation is broken by sampling randomly from the experience replay buffer.

Algorithm 1 shows the pseudocode of a basic DQN implementation with double Q-learning [52], which helps reducing the overestimation bias of conventional Q-learning by decoupling the selection of the action from its evaluation. We refer the reader to [53] to check for more extensions that further improve DQN.

Algorithm 1: DQN [51] Pseudocode

```
   Data: Learning rate $\alpha$, discount factor $\gamma$, exploration rate $\epsilon$, capacity of replay
         memory $D$, batch size $B$
   Result: Trained Q-network
1  Initialize Q-network with random weights $\theta$
2  Initialize target Q-network with weights $\theta^- = \theta$
3  Initialize replay memory $D$ with capacity $D$
4  for episode = 1 to $M$ do
5      Initialize state $s$
6      while episode is not terminated do
7          With probability $\epsilon$, select a random action $a$
8          Otherwise, select $a = \arg\max_{a'} Q(s, a'; \theta)$
9          Execute action $a$ in the environment
10         Observe reward $r$ and next state $s'$
11         Store transition $(s, a, r, s')$ in $D$
12         Sample random minibatch of transitions $(s_j, a_j, r_j, s'_j)$ from $D$
13         for each transition in minibatch do
14             If $s'_j$ is terminal, set $y_j = r_j$
15             Otherwise, set $y_j = r_j + \gamma \max_{a'} Q(s'_j, a'; \theta^-)$
16         end for
17         Perform a gradient descent step on $\big(y_j - Q(s_j, a_j; \theta)\big)^2$ with respect to $\theta$
18         Every $C$ steps, set $\theta^- = \theta$
19         Update state $s = s'$
20     end while
21 end for
```

2.2.2 Policy Gradient Methods

In model-free Policy Gradient Optimization methods [48] the agent learns an explicit policy representation π_θ with some parameters θ –typically a neural network. In most cases, during the training process, they involve learning as well a function approximator $V_\phi(s)$ of the state value function $V^{\pi_\theta}(s)$, defined as the expected discounted return from a given state s by following policy π_θ:

$$V^{\pi_\theta}(s) = \mathop{\mathbb{E}}_{\pi_\theta}\left[G_t | s_t = s\right] \tag{2.2.4}$$

This defines the so-called Actor-Critic family of Policy Gradient algorithms [48], where actions are selected from the function that estimates the policy (i.e., the actor), and the training of such policy is guided by the estimated value function to assess the consequences of the actions taken (i.e., the critic). The optimization of both set of parameters θ and ϕ is generally performed in an online fashion –i.e., trajectories used for the updates are always drawn from the latest version of the policy.

Among the great variety of Actor-Critic algorithms proposed in the literature [48], in this thesis we particularly implement a Proximal Policy Optimization (PPO) [54] method, which strikes a favorable balance between reliability, sample complexity and simplicity. In fact, PPO emerged as an alternative of Trust Region Policy Optimization (TRPO) methods [55], sharing most of their benefits while being much simpler to implement. We refer the reader to the original paper [54] for further details of the PPO motivation and its relation to TRPO.

In order to improve data efficiency, PPO uses multiple epochs of minibatch Stochastic Gradient Descent (SGD) to perform a policy update from a certain trajectory; in particular, this implies that there might be a difference between the current stochastic policy π_θ during the optimization process and the policy $\pi_{\theta_{old}}$ that generated the samples. PPO deals with this degree of "off-policyness" through importance sampling, so minibatch policy updates are performed by maximizing an objective of the form

$$L(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[\rho_t(\theta) \hat{A}_t \right], \tag{2.2.5}$$

being $\hat{A}_t = \hat{G}_t - V_\phi(s_t)$ the estimated advantage function at step t, which can be approximated through the critic's state value function (e.g. by means of Generalized Advantage Estimate (GAE) [56]).

However, without a constraint, the maximization of such objective would lead to an excessively large policy update [55]. Authors of PPO redefine it to penalize changes to the policy that move $\rho_t(\theta)$ away from 1:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(\rho_t(\theta)\hat{A}_t, \operatorname{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t) \right] \tag{2.2.6}$$

By introducing the hyperparameter ϵ, the term $\operatorname{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)$ keeps the probability ratio $\rho_t(\theta)$ in the interval $[1-\epsilon, 1+\epsilon]$. Finally, by taking the minimum of the the clipped and unclipped objective, PPO objective becomes a lower -i.e. pessimistic- bound of the unclipped objective.

Regarding the training of the critic, the value function error term $L^{VF}(\phi)$ is typically a squared-error loss between the prediction $V_\phi(s)$ and the actual cumulative reward obtained in the agent trajectory. Moreover, an extra entropy term $S[\pi_\theta]$ is usually considered to ensure sufficient exploration, thus obtaining a global PPO objective of the form

$$L^{PPO}(\theta, \phi) = \mathbb{E} \left[L^{CLIP}(\theta) - c_1 L^{VF}(\phi) - c_2 S[\pi_\theta] \right], \tag{2.2.7}$$

where c_1, c_2 are coefficients to weight critic and entropy losses, respectively.

2.3 Multi-Agent Reinforcement Learning

Contrary to a single-agent RL setting, in a MARL framework there is a set of agents $\mathcal{V}$ –interacting with a common environment– that have to learn how to cooperate to pursue a common goal. Such a setting is generally formulated as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [21] where, besides the global state space $\mathcal{S}$ and action space $\mathcal{A}$, it distinguishes local state and action spaces for every agent –i.e., $\mathcal{S}_v$ and $\mathcal{A}_v$ for $v \in \mathcal{V}$. At each time step t of an episode, each agent may choose an action $a_t^v \in \mathcal{A}_v$ based on local observations of the environment encoded in its current state $s_t^v \in \mathcal{S}_v$. Then, the environment produces individual rewards r_t^v (and/or a global one r_t), and it evolves to a next global state $s_{t+1} \in \mathcal{S}$ –i.e., each agent v turn into the following state $s_{t+1}^v \in \mathcal{S}_v$. Typically, a MARL system seeks for the optimal global policy by learning a set of local policies $\{\pi_{\theta_v}\}_{v \in \mathcal{V}}$. For doing so, most state-of-the-art MARL solutions implement traditional (single-agent) RL algorithms on each distributed agent, while incorporating some kind of cooperation mechanism between them [21]. The standard approach for obtaining a robust decentralized execution, however, is based on a centralized training where extra information can be used to guide agents' learning [57].

Chapter 3

Multi-Agent Graph Neural Network Optimization

In this chapter we introduce the main contribution of this thesis: our novel ML framework for distributed optimization in networked scenarios, which leverages Graph Neural Network (GNN) [26] and Multi-Agent Reinforcement Learning (MARL) [21] at its core. We name it MAGNNETO, standing for Multi-Agent Graph Neural Network Optimization.

The organization of the chapter is as follows: in the first section we revisit some related works, then we describe in detail the general pipeline of MAGNNETO, and finally we summarize the main contributions of the proposed method.

3.1 Related Work

The aim of this section is to review the related literature of the general problem of optimizing networked, graph-based environments, without focusing on a specific use case. In the following chapters, when describing the application of MAGNNETO to particular scenarios (Traffic Engineering (TE) in Chapter 4, Congestion Control (CC) in Datacenter Network (DCN)s in Chapter 5), the reader can also find the corresponding section that revisits in depth the most related works on that field.

Typically, the optimization of networked scenarios has been formulated as a combinatorial problem and tackled using classical methodologies such as Integer Linear Programming (ILP) [15, 16], Local Search (LS) [17], Constraint Programming (CP) [18] or CP [19], to name a few. However, all of these centralized methods are computationally expensive and do not scale well with the graph size; hence, despite their theoretical guarantees, they result in sub-optimal solutions if the execution time is not long enough. This also prevents their use in real-time applications, as the optimization process needs to be recomputed from scratch after every change in the network.

On the other hand, the design of decentralized architectures can hugely benefit from the distributed nature of the involved networks: it can potentially alleviate the scalability issues by naturally parallellizing the optimization process across computationally-enabled elements of the graph. In this regard, ML solutions, and in particular MARL-based methodologies [21], have attracted a lot of interest in recent years [22–25]. However, most of the proposed methods implement standard NNs to model the policies and either define greedy agents with barely cooperation between them, or rely on *ad hoc* communication mechanisms that are specifically defined for the particular use case.

In fact, to the best of our knowledge, before we published the first version of our method – [32], *Is Machine Learning Ready for Traffic Engineering Optimization?*– there were very few MARL frameworks designed to exploit the graph-based inductive biases of networked environments [27, 28], and the found references were purely AI-targeted papers focused on the general model formulation with no realistic applications in real-world scenarios.

Therefore, we claim that MAGNNETO is among the first architectures that embeds a GNN into a MARL setting to provide agents with topology-awareness and generalization capabilities while simplifying the cooperation mechanisms, and the first one that addresses the deployability challenges in real-world networked contexts (Internet Service Provider (ISP) Networks [32, 33], DCNs [34], Power Grids [58]). In next Section 3.3 we provide further details about the main contributions of MAGNNETO.

3.2 MAGNNETO Architecture

MAGNNETO internally models a networked environment as a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{V})$, with $\mathcal{N}$ and $\mathcal{E}$ representing the set of nodes and edges, respectively, and $\mathcal{V}$ acting for a set of agents that can control some of the graph entities (nodes or edges). Let $\mathcal{S}$ and $\mathcal{A}$ represent the global state and action spaces, respectively, defined as the joint and union of the respective agents' local spaces, $\mathcal{S} = \prod_{v\in\mathcal{V}} \mathcal{S}_v$ and $\mathcal{A} = \bigcup_{v\in\mathcal{V}} \mathcal{A}_v$. The theoretical framework of MAGNNETO allows to implement both Q-learning and PG methods, so for the sake of generalization let f_θ represent the global Reinforcement Learning (RL)-based function that is aimed to be learned –i.e., the global state-action value function Q_θ for the former, or the global policy π_θ for the latter.

One of the main features of MAGNNETO is that it makes all agents $v \in \mathcal{V}$ learn the global RL-based function approximator in a fully distributed fashion –i.e., all agents end up constructing and having access to the very same representation f_θ. In particular, and from a theoretical RL standpoint, this allows to formulate the problem within two different paradigms depending on the number of actions allowed at each time-step of the RL episode. On the one hand, imposing a single action per time-step enables to devise the problem as a time-homogeneous Markov Decision Process (MDP) of single-agent RL [48]. On the other hand, when letting several agents act simultaneously, it requires the more challenging

Decentralized Partially Observable Markov Decision Process (Dec-POMDP) formalization of standard MARL [21]. Note, however, that in practice the execution pipeline of MAGNNETO is exactly the same in both cases.

Another relevant feature of our design is that all agents $v \in \mathcal{V}$ are able to internally construct such global representation f_θ mainly through message communications with their direct neighboring agents $\mathcal{B}(v)$ and their local computations, no longer needing a centralized entity responsible for collecting and processing all the global information together. Such a decentralized, message-based generation of the global function is achieved by modeling the global function f_θ with a Message Passing Neural Network (MPNN), so that all agents $v \in \mathcal{V}$ deployed in the network are actually *replicas* of the MPNN modules (message, aggregation, update and readout functions) that perform regular message exchanges with their neighbors $\mathcal{B}(v)$ following the message passing iteration procedure of MPNNs; in particular, note that such *parameter sharing* implies that all agents share as well the same local state and action spaces. This reinterpretation of a MPNN as a set of copies of its internal modules is especially important due to the fact that in our approach we directly map the graph $\mathcal{G}$ to a real networked scenario, deploying copies of the MPNN modules along hardware devices in the network (e.g., routers) and making all message communications involved to actually go through the real network infrastructure. Hence, our proposed architecture naturally distributes the execution of the MPNN, and consequently is able to fully decentralize the execution of single-agent RL algorithms.

Algorithm 2 summarizes the resulting distributed pipeline. At each time-step t of an episode of length T, the MPNN-driven process of approximating the function $f_\theta(s_t, a_t)$ –where $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ refer to the global state and action at t– first constructs a meaningful hidden state h_v for each agent $v \in \mathcal{V}$. Each hidden state h_v basically depends on the hidden representations of the neighboring agents $\mathcal{B}(v)$, and its initialization h_v^0 is a function of the current agent state $s_v^t \in \mathcal{S}_v$, which is in turn based on some pre-defined internal agent features x_v^t. Those representations are shaped during K message-passing steps, where hidden states are iteratively propagated through the graph via messages between direct neighbors. In particular, successive hidden states h_v^k, where k accounts for the message-passing step, are computed by the message, aggregation and update functions of the MPNN, as previously described in Section 2.1.

Once agents generate their final hidden representation, a readout function –following the MPNN nomenclature– is applied to each agent to finally obtain the global function f_θ. Particularly, in our system the readout is divided into two steps: first, each agent $v \in \mathcal{V}$ implements a local readout that takes as input the final representation h_v^K, and outputs the final value -or a representation- of the global function f_θ over every possible action in the agent's space $\mathcal{A}_v$; for instance, this output could be the unnormalized log probability (i.e., logit) of the agent's actions in case of PG methods, or directly the q-value associated to each action when considering Q-learning algorithms. The second and last steps involve

Algorithm 2: MAGNNETO's execution pipeline.

```
Require: A graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ with a set of agents $\mathcal{V}$, MPNN trained parameters
         $\theta = \{\theta_i\}_{i \in \{m,a,u,r\}}$
Input: Initial graph configuration $X^0_{\mathcal{G}}$, episode length $T$, number of message passing
       steps $K$
1  Agents initialize their states $s^0_v$ based on $X^0_{\mathcal{G}}$
2  for $t \leftarrow 0$ to $T$ do
3      Agents initialize their hidden states $h^0_v \leftarrow (s^t_v, 0, \ldots, 0)$
4      for $k \leftarrow 0$ to $K$ do
5          Agents share their current hidden state $h^k_v$ to neighboring agents $\mathcal{B}(v)$
6          Agents process the received messages $M^k_v \leftarrow a_{\theta_a}(\{m_{\theta_m}(h^k_v, h^k_\mu)\}_{\mu \in \mathcal{B}(v)})$
7          Agents update their hidden state $h^{k+1}_v \leftarrow u(h^k_v, M^k_v)$
8      end for
9      Agents partially evaluate the RL function $f_\theta$ over their own actions
         $\{f_\theta(s_t, a)\}_{a \in \mathcal{A}_v} \leftarrow r_{\theta_r}(h^K_v)$
10     Agents receive the partial evaluations of $f_\theta$ of the rest of agents and build the
         global representation $f_\theta \leftarrow \{f_\theta(s_t, a)\}_{a \in \mathcal{A}}$
11     Agents select the same set of actions $A_t$ according to $f_\theta$
12     Agents whose action was selected execute it, and the environment updates the
         graph configuration $X^{t+1}_{\mathcal{G}}$
13     Agents update their states $s^{t+1}_v$ based on $X^{t+1}_{\mathcal{G}}$
14 end for
Output: New graph configuration $X^*_{\mathcal{G}}$ that optimizes some pre-defined objective or
        metric
```

a communication layer that propagates such individual outputs to the rest of the agents, so that all of them can internally construct the global representation of f_θ for the overall network state $s_t = \prod_{v \in \mathcal{V}} s^t_v$ and all possible actions $\bigcup_{v \in \mathcal{V}} \{a_{v,0}, a_{v,1}, \ldots, a_{v,i}\}$, with $i \in \mathbb{N} \backslash \{0\}$ the number of actions of local agent spaces $\mathcal{A}_v$. Finally, to ensure that all distributed agents sample the same actions when f_θ encodes a distribution, they are provided with the same probabilistic seed before initiating the process. Consequently, only agents whose action has been selected do execute an action at each time-step t. Note that actions are not actually applied over the network configuration until the whole optimization process finishes.

3.3 Main Features and Contributions

Finally, this section summarizes the main contributions and features of the proposed MAGNNETO's pipeline:

Adaptation to Different Networked Scenarios: By design, MAGNNETO can easily be adapted to any networked scenario where some of the node elements of its graph-based representation (e.g. routers in computer networks, power stations in power grids) have computation capabilities, and there exist physical or wireless links between them that can naturally transmit the communication messages between neighboring agents involved in the MPNN-based optimization process. The critical step is to define an appropiate RL environment (episode definition, state and action spaces, reward function) for the MARL setting to properly learn to optimize the desired network metric.

Fully Decentralized Architecture: In contrast to traditional centralized optimization schemes, the decentralized design of MAGNNETO naturally distributes and parallelizes the optimization execution across the network. As we will show in our considered applications, this allows our method to run significantly faster than centralized solutions while achieving comparable state-of-the-art performance.

Generalization over Unseen Networks: A common downside of current ML-based solutions applied to networked environments is their limited performance when operating in different networks to those seen during training, which is commonly referred to as lack of *generalization* [39]. Without generalization, training *must* be done at the same network where the ML-based solution is expected to operate. Hence, from a practical standpoint generalization is a crucial aspect, as training directly in networks in production is typically unfeasible. MAGNNETO implements internally a GNN, which introduces proper learning biases to generalize across networks of different sizes and structures [39].

No Need of Network Upgrades: As stated before in the first point, the only requirement for a networked scenario to be compatible with MAGNNETO are basic computation capabilities on its nodes, which can be taken for granted in most –if not all– of today's infrastructures. In particular, note that no new hardware would be required for its implementation and deployment, i.e. it is totally compatible with current and legacy equipment. Each agent, as a replica of the inner modules of the GNN, does not necessarily need specific AI accelerators to perform its computations.

Flexibility to Define the RL Algorithm: As mentioned in previous Section, MAGNNETO is compatible with both Q-learning and Policy Gradient RL methods. As we will see in next chapters, this is an important feature since the particularities of the considered use case –specially the definition of the action space– can result in different needs regarding those types of RL pipelines. For instance, for the application of MAGNNETO to CC in DCNs (Chapter 5) Deep Q-learning Network (DQN) showed more stability and robustness than Proximal Policy Optimization (PPO) when navigating among the large action space, whereas in the TE use case (Chapter 4) PPO clearly outperformed the results obtained by Q-learning methods.

Applications

Chapter 4

MAGNNETO-TE: Traffic Engineering in ISP Networks

During the last decade, the networking community has devoted significant efforts to build efficient solutions for automated network control, pursuing the ultimate goal of achieving the long-desired *self-driving networks* [59,60]. In this vein, Machine Learning (ML) is considered as a promising technique for producing efficient tools for autonomous networking [61, 62].

In this chapter, we revisit a fundamental networking problem: Traffic Engineering (TE) optimization [63, 64]. TE is among the most common operation tasks in today's Internet Service Provider (ISP) networks. Here, the classical optimization goal is to minimize network congestion, which is typically achieved by minimizing the maximum link utilization in the network [16–18, 65, 66]. Given the relevance of this problem, we have witnessed a plethora of proposals approaching this problem from different angles, such as optimizing the configuration of widely deployed link-state protocols (e.g., Open Shortest Path First (OSPF) [67]), making fine-grained flow-based routing, or re-routing traffic across overlay networks [68, 69].

Likewise, for the last years the networking community has focused on developing effective ML-based solutions for TE. In particular, many works propose the use of Reinforcement Learning (RL) for efficient TE optimization (e.g., [23, 70–72]). However, at the time of the start of our research, no ML-based proposal had succeeded to replace long-established TE solutions; indeed, the best performing TE optimizers to that date were based on traditional optimization algorithms, such as Constraint Programming (CP) [18], Local Search (LS) [17], or Linear Programming [15, 16].

In this chapter, we introduce MAGNNETO-TE, the adaptation of our general MAGNNETO framework to TE. In the proposed algorithm, a RL-based agent is deployed on each router. Similarly to standard intradomain routing protocols (e.g., OSPF), MAGNNETO's agents exchange information with their neighbors in a distributed manner. In particular, agents communicate via a neural network-driven message passing mechanism, and learn how to cooperate to pursue a common optimization goal. As a result, the proposed framework is fully distributed, and agents learn how to effectively communicate to perform intradomain TE optimization, i.e. to minimize the maximum link utilization in the network.

Remark: In this chapter, we will interchangeably employ the acronyms MAGNNETO-TE and MAGNNETO to denote the adaptation of the framework to TE. Regarding the overarching architecture introduced in Chapter 3, we shall consistently address it as the general MAGNNETO framework.

More in detail, this adaptation of MAGNNETO presents the following main contributions to TE research:

Top performance with very low execution times: We compare MAGNNETO-TE against a curated set of well-established TE solutions: SRLS [17], DEFO [18] and TabuIG-PWO [16]. These solutions implement mature optimization techniques on top of expert knowledge. As a result, they are able to achieve close-to-optimal performance in large-scale networks within minutes [73]. Our results show that MAGNNETO-TE achieves comparable performance to these state-of-the-art TE solutions, while being significantly faster. In fact, when enabling several simultaneous actions in our framework, it runs up to three orders of magnitude faster than the baseline optimizers (sub-second vs. minutes) in networks with 100+ nodes. The reason for this is the fully decentralized architecture of MAGNNETO, which naturally distributes and parallelizes the execution across the network.

Generalization over unseen networks: A common downside of current ML-based solutions applied to networking is their limited performance when operating in different networks to those seen during training, which is commonly referred to as lack of *generalization* [39]. Without generalization, training *must* be done at the same network where the ML-based solution is expected to operate. Hence, from a practical standpoint generalization is a crucial aspect, as training directly in networks in production is typically unfeasible. MAGNNETO-TE implements internally a Graph Neural Network (GNN), which introduces proper learning biases to generalize across networks of different sizes and structures [39]. In our evaluation, we train MAGNNETO in two different networks, and test its performance and speed on 75 real-world topologies from the Internet Topology Zoo [74] not seen before. Our results show that in such scenarios, MAGNNETO still achieves comparable performance to state-of-the-art TE optimizers, while being significantly faster.

No need for overlay technologies: Recent TE optimizers rely on novel overlay technologies to achieve their optimization goals [17, 18]. By leveraging Segment Routing (SR) [75] these solutions are able to use arbitrary overlay paths that are not routed via the standard OSPF weights. This allows to extend the routing space to a source-destination granularity and –as shown in the literature– it renders outstanding results. However, in this chapter we show that comparable performance is achievable by using only standard

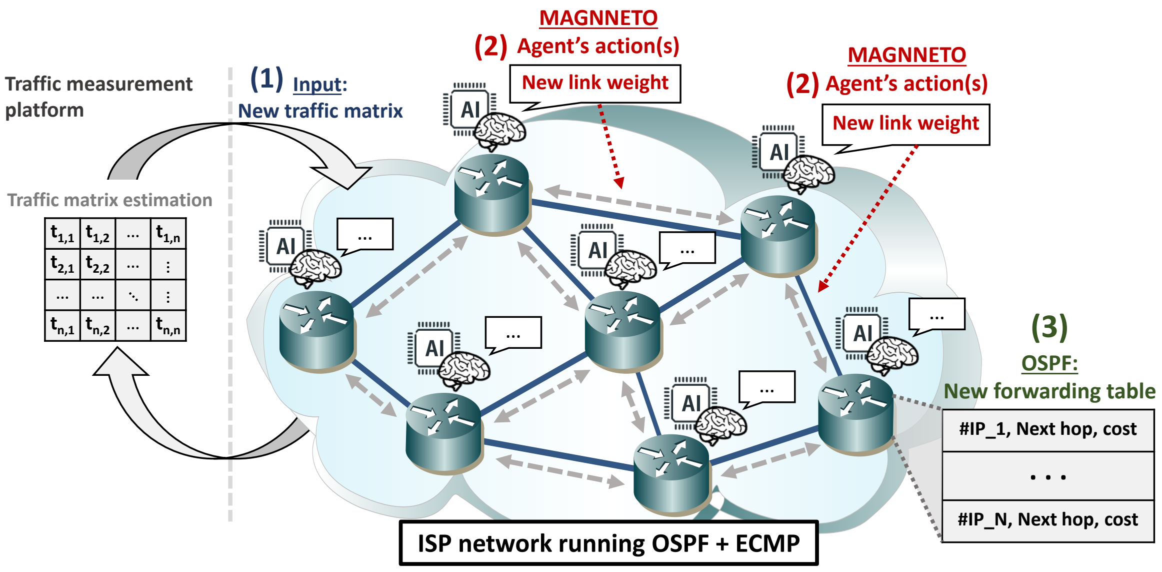


Figure 4.1: Intradomain Traffic Engineering optimization with MAGNNETO.

destination-based OSPF routing. Indeed, MAGNNETO-TE is fully compliant with current OSPF-based networks, and does not require the use of any overlay technology.

The remainder of this chapter is as follows. Section 4.1 describes the TE scenario where we deploy the proposed ML-based system. Section 4.2 describes how we adapt the general MAGNNETO framework to perform intradomain TE optimization. Section 4.3 presents the first exploratory results we obtained, when MAGNNETO's architecture still had some limitations. In Section 4.4 we extend previous evaluation with the full MAGNNETO version, and consider a larger set of state-of-the-art TE optimizers as well as many more network topologies. Section 4.5 summarizes the main existing works related to this use case, and lastly Section 4.6 concludes the chapter.

4.1 Network Scenario

This section describes the intradomain TE scenario over which MAGNNETO-TE operates, where network traffic is measured and routed to minimize network congestion. Typically, ISP networks run a link-state Interior Gateway Protocol (IGP), such as Open Shortest Path First (OSPF) [67], that chooses paths using the Dijkstra's algorithm over some pre-defined link weights.

There exists a wide range of architectures and algorithms for TE in the literature [76]. Network operators commonly use commercial tools [77, 78] to fine-tune link weights. However, other mechanisms propose to add extra routing entries [79] or end-to-end tunnels (e.g., RSVP-TE [80]) to perform source-destination routing, thus expanding the solution space.

MAGNNETO is a fully distributed framework that interfaces with standard OSPF, by optimizing the link weights used by such protocol. As a result, it does not require any changes to OSPF and it can be implemented with a software update on the routers where it is deployed. In this context, relying on well-known link-state routing protocols, such as OSPF, offers the advantage that the network is easier to manage compared to finer-grained alternatives, such as flow-based routing [81].

Figure 4.1 illustrates the general operational workflow of MAGNNETO-TE:

1) Traffic Measurement: First, a traffic measurement platform deployed over the network identifies a new Traffic Matrix (TM). This new TM is communicated to all participating routers (Fig. 4.1, step 1), which upon reception will start the next step and optimize the routing for this TM. We leave out of the scope of this section the details of this process, as TM estimation is an extensive research field with many established proposals. For instance, this process can be done periodically (e.g., each 5-10 minutes as in [16]), where the TM is first estimated and then optimized. Some proposals trigger the optimization process when a relevant change is detected in the TM [82], while others use prediction techniques to optimize it in advance [83]. Also, some real-world operators make estimates considering their customers' subscriptions and operate based on a static TM. Our proposal is flexible and can operate with any of these approaches.

2) MAGNNETO-TE optimization: Once routers receive the new TM, the distributed RL-based agents of MAGNNETO start the TE optimization process, which eventually computes the per-link weights that optimize OSPF routing in the subsequent step (Fig. 4.1, step 2). Particularly, we set the goal to minimize the maximum link load (*MinMaxLoad*), which is a classic TE goal in carrier-grade networks [18, 65, 66]. This problem is known to be NP-hard, and even good settings of the weights can deviate significantly from the optimal configuration [66, 81]. Our Multi-Agent Reinforcement Learning (MARL) optimization system is built using a distributed GNN that exchanges messages over the physical network topology. Messages are sent between routers and their directly attached neighbors. The content of such messages are *hidden states* that are produced and consumed by artificial neural networks and do not have a human-understandable *meaning*. The GNN makes several message iterations and, during this phase, local configuration of the router remains unchanged, thus having no impact on the current traffic. More details about the inner workings, performance, communication overhead, and computational cost can be found in Sections 4.2-4.4.

3) OSPF convergence: Lastly, the standard OSPF convergence process is executed taking into account the new per-link weights computed by MAGNNETO-TE. Specifically, each agent has computed the optimal weigths for its locally attached links. For OSPF to recompute the new forwarding tables, it needs to broadcast the new link weights; this is done using the standard OSPF Link-State Advertisement (LSA) [67]. Once the routers have an identical view of the network, they compute locally their new forwarding tables (Fig. 4.1, step 3), and traffic is routed following the optimization goal. Convergence time of OSPF is a well-studied subject. For instance, routing tables can converge in the order of a few seconds in networks with thousands of links [84].

4.2 MAGNNETO-TE

In this section we describe the particular adaptations of the general MAGNNETO framework when applying it to the intradomain TE scenario described in Section 4.1. Moreover, we provide some details about the training pipeline of our models.

4.2.1 General Setting

A straightforward approach to map the graph $\mathcal{G}$ of the described MAGNNETO framework (Chapter 3) to a computer network infrastructure is to associate the nodes $\mathcal{N}$ to hardware devices (e.g., router, switches) and the edges $\mathcal{E}$ to the physical links of the network. Regarding the set of agents $\mathcal{V}$, they can be identified either with the set of nodes, so that they individually control a hardware device, or with the set of edges by controlling some configuration parameters of a link connecting two devices.

In the intradomain TE problem, the goal is to learn the set of link weights $\mathcal{W} = \{w_e\}_{e\in\mathcal{E}}$ that minimizes the maximum link utilization for a certain traffic matrix TM. Hence, we adapt MAGNNETO-TE so that each agent controls a link (i.e., $\mathcal{V}\widehat{=}\mathcal{E}$) and can modify its weight w_e; in fact, in order to make the notation simpler, from now on we will refer to each agent $v \in \mathcal{V}$ as the edge $e \in \mathcal{E}$ it represents. We also note that:

- computer networks are commonly represented as directed graphs with links in both directions, so for each directed link $e = (n_e^{src}, n_e^{dst}) \in \mathcal{E}$, with $n_e^{src}, n_e^{dst} \in \mathcal{N}$, we define its neighbor as the set $\mathcal{B}(e)$ of edges whose source node coincides with the destination node of e, i.e. $\mathcal{B}(e) = \{e' \in \mathcal{E} | n_{e'}^{src} = n_e^{dst}\}$. In other words, edges in $\mathcal{B}(e)$ are those links that can potentially receive traffic from link e.
- in practice, link-based agents $e \in \mathcal{E}$ would be deployed and executed in their adjacent source (n_e^{src}) or destination (n_e^{dst}) hardware device.

Furthermore, we implement a well-known Actor-Critic method named Proximal Policy Optimization (PPO) [54], which offers a favorable balance between reliability, sample complexity, and simplicity. Consequently, in this case the global function f_θ of the framework (see Section 3) is the global policy π_θ of the actor. Regarding the critic's design, more information can be found in Section 4.2.3.

4.2.2 Adapting MAGNNETO to TE

Having clear the general configuration of our MAGNNETO-TE implementation, now we will further describe its operation when dealing with the intradomain TE objective. To do so, let us reinterpret each of the main fundamental elements introduced earlier from a TE perspective:

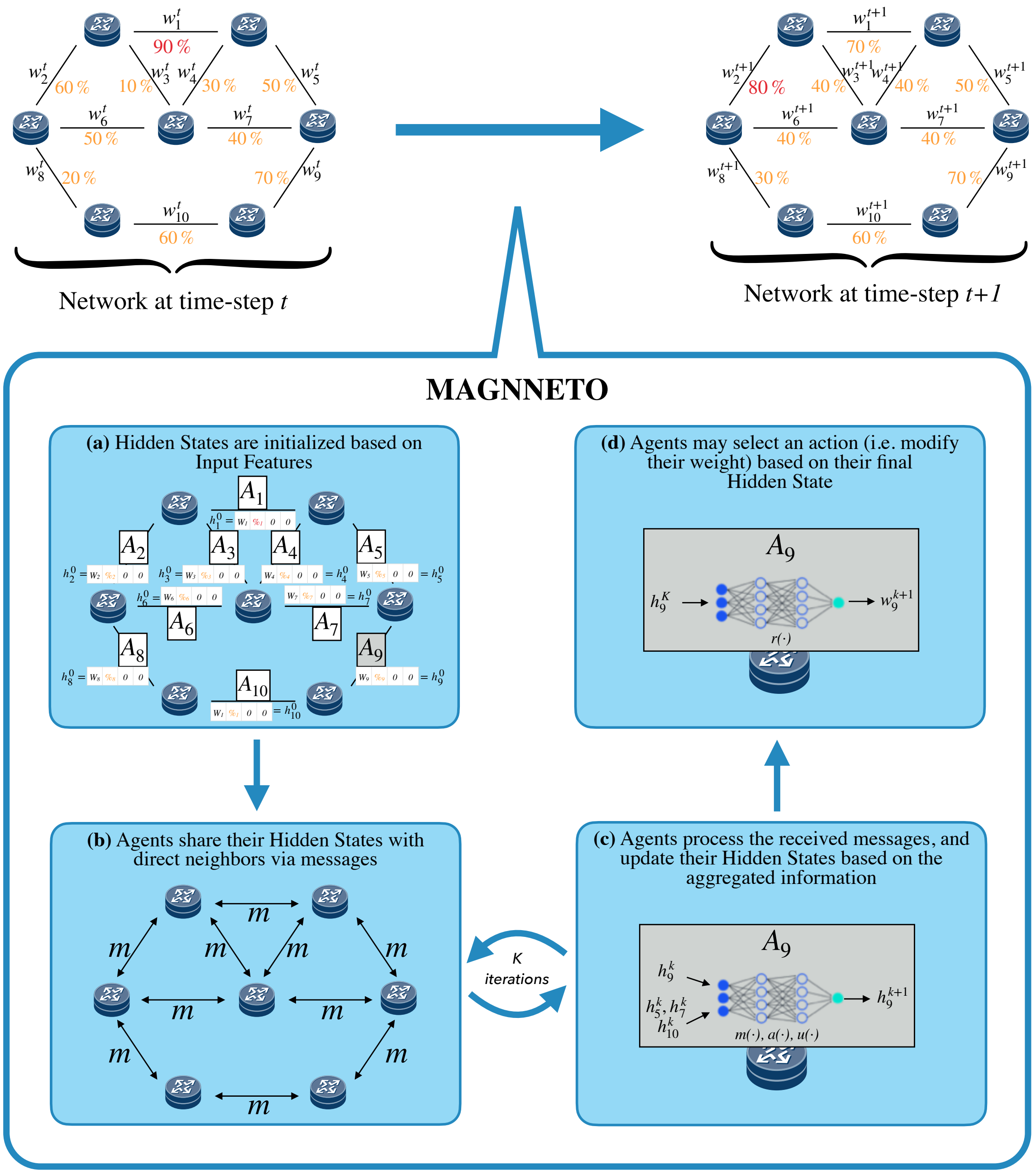

Network at time-step t
Network at time-step t+1
MAGNNETO
(a) Hidden States are initialized based on Input Features
(b) Agents share their Hidden States with direct neighbors via messages
(c) Agents process the received messages, and update their Hidden States based on the aggregated information
(d) Agents may select an action (i.e. modify their weight) based on their final Hidden State
K iterations
$r(\cdot)$
$m(\cdot), a(\cdot), u(\cdot)$

Figure 4.2: Description of the message passing and action selection process of MAGNNETO-TE at a certain time-step t of an episode. For simplicity, visual representations of steps (c) and (d) are focused on a single agent (A_9); however, note that the same procedure is executed in parallel in all link-based agents.

Environment

We consider episodes of a fixed number of time-steps T. At the beginning of each episode, the environment provides with a set of traffic demands between all source-destination pairs (i.e., an estimated traffic matrix [16]). Each link $e \in \mathcal{E}$ has an associated capacity c_e, and it is initialized with a certain link weight w_e^0. These link weights are in turn used to compute the routers' forwarding tables, using the standard Dijkstra's algorithm. Each agent $v_e \in \mathcal{V}$ has access to its associated link features, which in our case are the current weight, its capacity, the estimated traffic matrix and the weights of the other links. This can be achieved with standard procedures in OSPF-based environments (see Sec. 4.1).

State Space and Message Passing

At each time-step t of an episode, each link-based agent $v_e \in \mathcal{V}$, feeds its Message Passing Neural Network (MPNN) module with its input features x_e^t to generate its respective initial hidden state h_e^0 (Figure 4.2.a). In particular, agents consider as input features the current weight w_e^t and the utilization u_e^t $[0, 1]$ of the link, and construct their initial link hidden representations h_e^0 as a fixed-size vector where the first two components are the input features and the rest is zero-padded. Note that the link utilization can be easily computed by the agent with the information of the estimated traffic matrix and the global link weights locally maintained. Then, the algorithm performs K message-passing steps (Figures 4.2.b and 4.2.c). At each step k, the algorithm is executed in a distributed fashion over all the links of the network. Particularly, each link-based agent $e \in \mathcal{E}$ receives the hidden states of its neighboring agents $\mathcal{B}(e)$, and combines them individually with its own state h_e^k through the *message* function (a fully-connected NN). Then, all these outputs are gathered according to the *aggregation* function –in our case an element-wise min and max operations– producing the combination M_e^k. Afterwards, another fully-connected NN is used as the *update* function, which combines the link's hidden state h_e^k with the new aggregated information M_e^k, and produces a new hidden state representation for that link (h_e^{k+1}). As mentioned above, this process is repeated K times, leading to some final link hidden state representations h_e^K.

Action Space

In our implementation, each agent $e \in \mathcal{E}$ can only take a single action: to increase its link weight w_e in one unit. In particular, the agent's action selection (Figure 4.2.d) is done as follows: first, every agent applies a local readout function –implemented with a fully-connected NN– to its final hidden state h_e^K, from which it obtains the global logit estimate of choosing its action (i.e., increase its link weight) over the actions of the other agents. Then, as previously described in Section 3.2, these logits are shared among agents in the network, so that each of them can construct the global policy distribution π_θ. By sharing the same probabilistic seed, all agents sample locally the same set of actions A_t. Finally, agents whose action has been selected increase by one unit the weight of their associated link in its internal global state copy, which is then used to compute the new link utilization u_e^{t+1} under the new weight setting, as well as to initialize its hidden state representation in the next time-step $t + 1$.

Reward Function

During training, a reward function is computed at each step t of the optimization episode. In our case, given our optimization goal we directly define the reward r_t as the difference of the global maximum link utilization between steps t and $t + 1$. Note that this reward can be computed locally at each agent from its global state copy, which is incrementally updated with the new actions applied at each time-step.

4.2.3 Training Details

The training procedure highly depends on the type of RL algorithm chosen. In our particular implementation, given that we considered an Actor-Critic method (PPO), the objective at training is to optimize the parameters $\{\theta, \phi\}$ so that:

- the previously described GNN-based actor π_θ becomes a good estimator of the optimal global policy;
- the critic V_ϕ learns to approximate the state value function of any global state.

As commented in Section 2.2.2, the goal of the critic is to guide the learning process of the actor; it is no longer needed at execution time. Therefore, taking V_ϕ with a centralized design would have no impact on the distributed nature of MAGNNETO-TE.

In fact, following the standard approach of MARL systems [57], the training of MAGNNETO is performed in a centralized fashion, and such centrality precisely comes from the critic's model. In particular, we have implemented V_ϕ as another link-based MPNN, similar to the actor but with a centralized readout that takes as inputs all link hidden states in and outputs the value function estimate. We also considered a MPNN-based critic to exploit the relational reasoning provided by GNNs; however, note that any other alternative design might be valid as well.

At a high level, the offline training pipeline is as follows. First, an episode of length T is generated by following the current policy π_θ, while at the same time the critic's value function V_ϕ evaluates each visited global state; this defines a trajectory $\{s_t, a_t, r_t, p_t, V_t, s_{t+1}\}_{t=0}^{T-1}$, where $p_t = \pi_\theta(a_t|s_t)$ and $V_t := V_\phi(s_t)$. When the episode ends, this trajectory is used to update the model parameters –through several epochs of minibatch Stochastic Gradient Descent– by maximizing the global PPO objective $L^{PPO}(\theta, \phi)$ described in [54]. The same process of generating episodes and updating the model is repeated for a fixed number of iterations to guarantee convergence.

4.3 Exploratory Evaluation

In our first work about MAGNNETO-TE [32] we raised an open question –*Is ML ready for Traffic Engineering optimization?*– and our goal was to discuss whether state-of-the-art ML techniques are mature enough to outperform traditional TE solutions; to this end, we presented a first version of MAGNNETO framework for TE optimization. This section describes the exploratory evaluation performed on that work, where MAGNNETO had still the limitation of of being formulated over a classical Markov Decision Process (MDP) setting, i.e. where agents must take actions sequentially in a synchronized manner. In Section 4.4 we show the extended evaluation performed in our second version of MAGNNETO-TE, which supports simultaneous actions at each RL optimization step.

More in detail, in this section we present a first set of experiments –over real-world network topologies– to evaluate the single-action MAGNNETO-TE architecture, and particularly focus on comparing the proposed solution with DEFO [18] –which is arguably among the best performing and most advanced TE solutions available at the time of our research [73].

4.3.1 Experimental Setup

Along this evaluation section, we consider three real-world network topologies for training and evaluation of our model: 42-link NSFNet, 54-link GBN, and 72-link GEANT2 [85]. The length $\boldsymbol{T}$ of the training and evaluation episodes is pre-defined, and it varies from 100 to 200 steps, depending on the network topology size (see more details later in Sec. 4.3.6). At the beginning of each episode, the link weights are randomly selected as an integer in the range $[1, 4]$, so our system is evaluated over a wide variety of scenarios with random routing initializations. From that point on, at each step of an episode a single agent can modify its weight by increasing it in one unit, thus chaining the selected actions on the T time-steps of an episode.

Taking [86] as a reference for defining the hyperparameters' values of the solution, we ran several grid searches to appropriately fine-tune the model. The implemented optimizer is Adam with a learning rate of $3 \cdot 10^{-4}$, β=0.9, and ϵ=0.9. Regarding the PPO setting, the number of epochs for each training episode is set to 3 with batches of size 25, the discount factor γ is set to 0.97, and the clipping parameter to 0.25. We implement the Generalized Advantage Estimate (GAE), to estimate the advantage function with λ=0.9. In addition, we multiply the critic loss by a factor of 0.5, and we implement an entropy loss weighted by a factor of 0.001. Finally, links' hidden states $\boldsymbol{h_e}$ are encoded as 16-element vectors, and in each MPNN forward propagation $\boldsymbol{K}$=8 message passing steps are executed.

We consider two different traffic profiles: *(i)* uniform distribution of source-destination traffic demands, and *(ii)* traffic distributions following a gravity model [87], which produces more realistic Internet traffic matrices. For each set of experiments, the training process of our MARL+GNN system took about 24 hours running in a machine with a single CPU of 2.20 GHz (~1M training steps).

4.3.2 Baselines

This section describes the baselines we use to benchmark our MARL+GNN system in our experiments. We particularly consider two well-known TE alternatives:

- Default OSPF: We consider the routing configuration obtained by applying the OSPF protocol with the common assumption that link weights are inversely proportional to their capacities. We consider traffic splitting over multiple paths (OSPF with Equal Cost Multi-Path (ECMP)), which is a standard recommended best practice.

- Declarative and Expressive Forwarding Optimizer (DEFO) [18]: A centralized network optimizer that translates high-level goals of operators into network configurations in real-time (in the order of minutes). DEFO starts from a routing configuration already optimized with a commercial TE tool [77], and it uses CP [88] and SR [75] to further optimize it. To this end, DEFO reroutes traffic paths through a sequence of middlepoints, spreading their traffic over multiple ECMP paths. DEFO obtains close-to-optimal performance considering several network optimization goals, one of them being our intradomain TE goal of minimizing the most loaded link. We use the code publicly shared by the authors of DEFO[1]. For the sake of comparison, we also use OSPF-ECMP in the evaluation of our system (MARL+GNN), although it can also operate in scenarios without ECMP support.

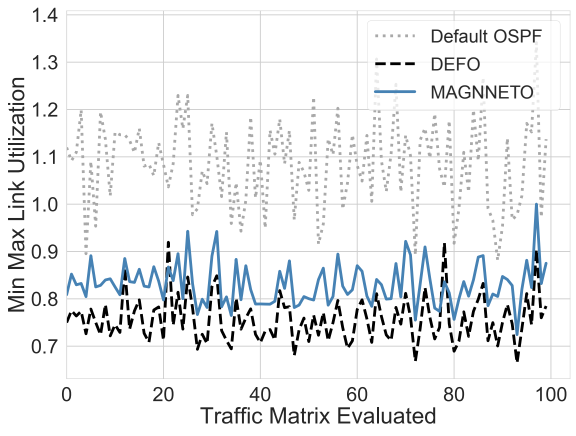
4.3.3 Performance Evaluation over different Traffic Matrices

In this subsection we present the results of our first experiment, which evaluates the performance of our proposed MARL solution over traffic matrices that have not been seen during the training process. More in detail, we consider a fixed network topology and a set of traffic matrices; then our model is trained in that single topology using a subset of traffic matrices, and finally the trained system is evaluated over a different set with unseen traffic.

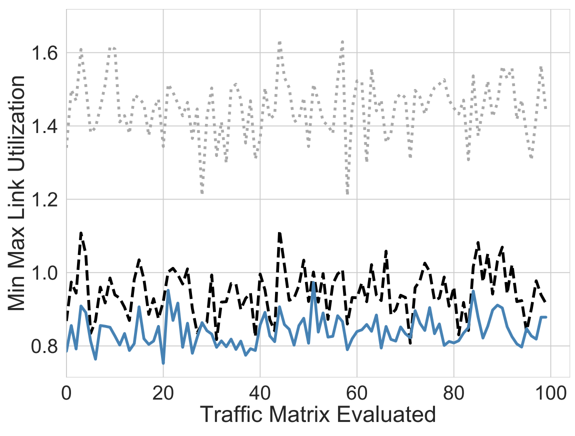
In particular, we analyze two different traffic profiles (uniform and gravity model), each of them in two network topologies (NSFNet and GEANT2). In total we run four independent experiments, one for each combination of traffic profile and topology. At each experiment, we stop the training when the system has observed around 100 different traffic matrices (TM), and the model is evaluated over 100 new TMs. During training, TMs change every 50 training episodes.

Figure 4.3 shows the evaluation result considering a uniform traffic profile. For the sake of readability, these plots show both the raw Minimum Maximum Link Utilization values

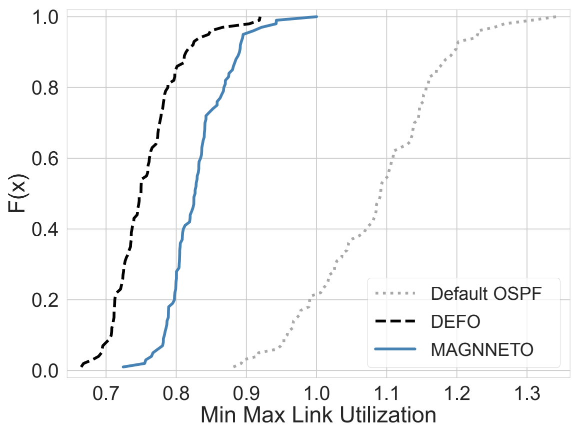
[1]`https://sites.uclouvain.be/defo/`



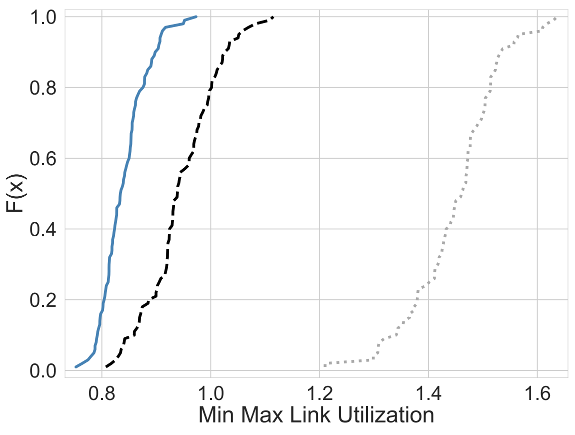
(a) *MinMaxLoad* NSFNet



(b) *MinMaxLoad* GEANT2



(c) CDF NSFNet



(d) CDF GEANT2

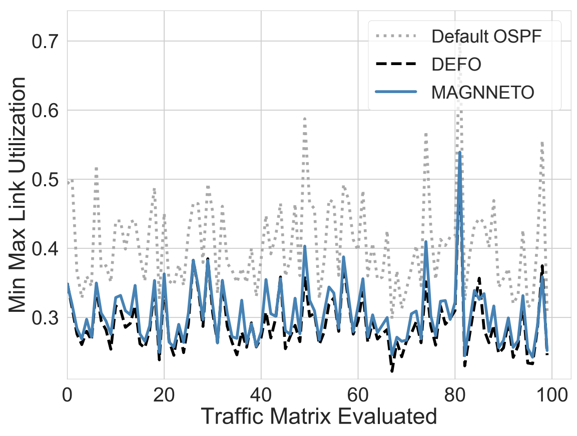
Figure 4.3: Evaluation results of Minimum Maximum Link Utilization with uniform traffic profiles in the NSFNet and GEANT2 network topologies. The evaluation is done over 100 traffic matrices unseen during training.

obtained for each TM, and the Cumulative Distribution Function (CDF) of these results. In this case, we can observe that our proposed MARL+GNN solution performs significantly better than default OSPF in both topologies (on average, ≈23% better in NSFNet and 42% in GEANT2) and stays near to the close-to-optimal solutions produced by DEFO algorithm (in GEANT2, it even improves it by 11%).

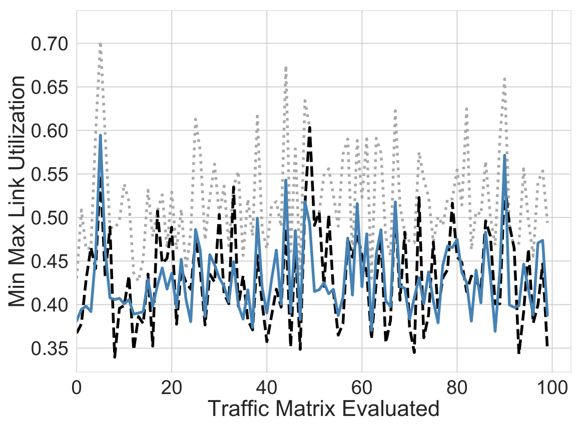
Analogously, Figure 4.4 presents the evaluation results in scenarios with the gravity traffic profile. Again, our proposed MARL+GNN solution outperforms default OSPF in both topologies (on average, ~25% better in NSFNet and 17% in GEANT2) and attains a comparable performance to DEFO.

4.3.4 Generalization over other Network Topologies

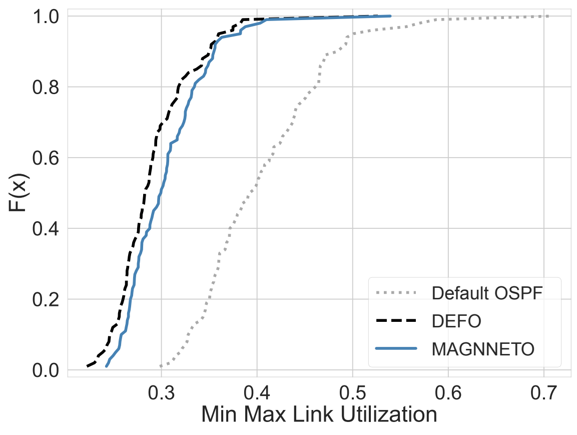
While traditional TE optimizers are typically designed to operate on arbitrary networks, current state-of-the-art ML-based solutions for TE suffer from a lack of topology



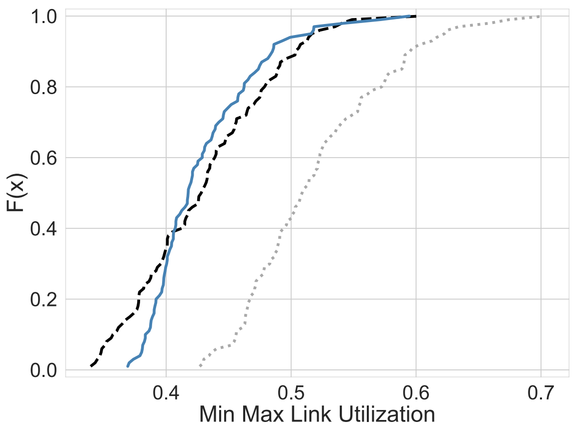
(a) *MinMaxLoad* NSFNet



(b) *MinMaxLoad* GEANT2

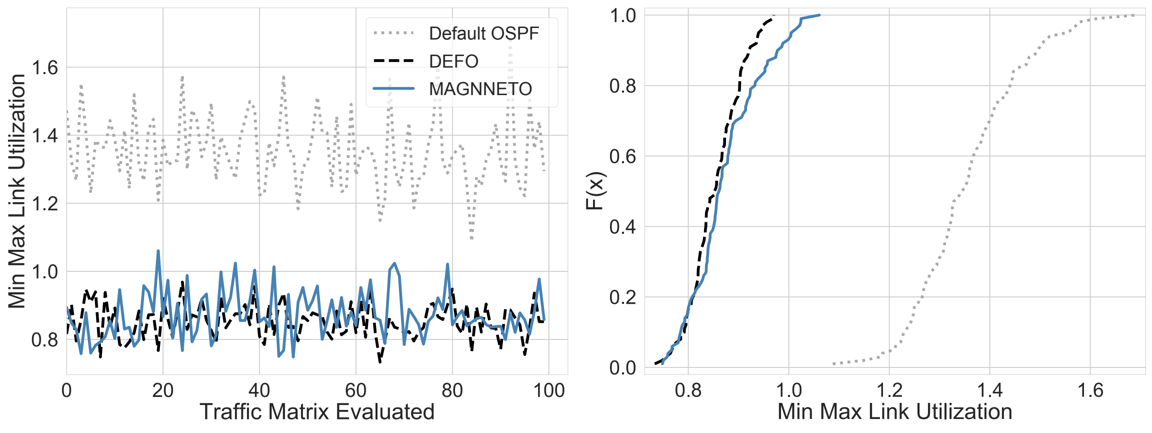


(c) CDF NSFNet



(d) CDF GEANT2

Figure 4.4: Evaluation results of Minimum Maximum Link Utilization with gravity-based traffic profiles in the NSFNet and GEANT2 network topologies. The evaluation is done over 100 traffic matrices unseen during training.



(a) *MinMaxLoad* GBN (b) CDF GBN

Figure 4.5: Evaluation results of Minimum Maximum Link Utilization for 100 different configurations in GBN after training the model using exclusively with samples of NSFNet and GEANT2.

generalization, partly explained by the fixed-size input scheme of most ML models (fully-connected NNs, convolutional NNs). That is, previous ML solutions could only operate on those toplogies seen during the training phase. Therefore, achieving generalization over different topologies is an essential step towards the versatility of state-of-the-art classical TE methods.

Given that our distributed GNN-based proposal naturally allows variable-size network scenarios, as well as relational reasoning [89,90], we are particularly interested in evaluating the generalization potential of our MARL+GNN solution over other networks not considered in training. For these experiments, we train our model in both NSFNet and GEANT2 topologies, and then evaluate it in a never-seen network (GBN). In this case, we stop the training when the system observes a total of 100 TMs –alternating NSFNet and GEANT2 instances every 50 training episodes– and evaluate it over 100 TMs in GBN.

Figure 4.5 presents the evaluation results of this experiment, showing the Minimum Maximum Link Utilization values obtained at each sample, as well as the CDF of these results. Here we can observe that the proposed solution significantly outperforms default OSPF (35% better on average) and it is very close -only within a 2% difference- to DEFO.

4.3.5 Robustness against Link Failures

The ability to generalize over different network topologies opens the door to addressing other uses cases that could not be solved with previous ML-based solutions. For example, in this section we assess how our solution performs when the network experiences link failures, which inevitably result in changes in the topology. To this end, we design the following experiment: given a traffic matrix and a topology, our model previously trained in Section 4.3.4 is applied in networks with increasing number of random link failures –up to

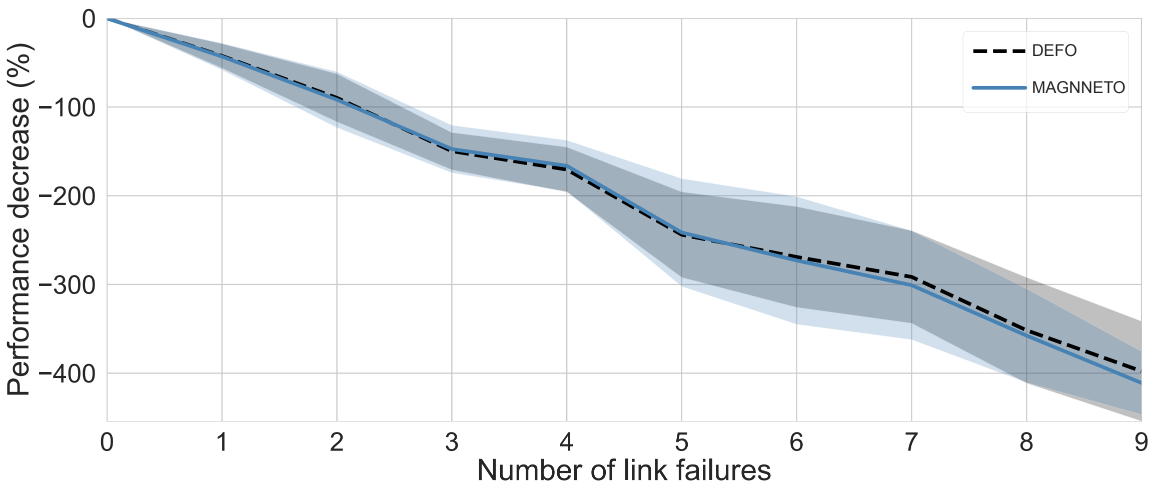


Figure 4.6: Performance degradation with increasing link failures for our NSFNet+GEANT2 model (applied to GBN), and DEFO. The plot shows the mean and standard deviation for 5 different TMs; for each TM we average the results on 10 scenarios with n random link failures.

a maximum of 9 failures. We repeat this experiment 10 times for a given number of failures n, exploring at each iteration different combinations of link failures.

Figure 4.6 shows the mean and standard deviation of the performance degradation –w.r.t. the original network scenario with all the links– over 5 different traffic matrices, using the model trained exclusively in NSFNet and GEANT2 (Sec. 4.3.4) and applying it over the GBN network topology. These results are compared against DEFO, which is evaluated under the same conditions (i.e., same TMs and network scenarios). As we can observe, the performance decays gracefully as the number of removed links increases, showing an almost identical behavior to that of the state-of-the-art DEFO technique.

	NSFNet	**GBN**	**GEANT2**	**SYNT500**	**SYNT1000**
Episode Length	100	150	200	5,250	9,600
Execution Time (s)	$9.98 \cdot 10^{-2}$	$1.33 \cdot 10^{-1}$	$2.12 \cdot 10^{-1}$	8.40	19.2
Link Overhead* (MB/s)	1.20	1.32	1.20	1.60	1.41

*It includes a 20% extra cost per message considering headers and metadata.

Table 4.1: Execution cost: Execution time and average MPNN-based link overhead. Applied to variable-sized network topologies, and assuming that hidden states are encoded as 16-element vectors of floats, and each Message Passing runs K=8 steps.

4.3.6 Performance vs. Message Passing Iterations

Previous experiments have shown that the proposed solution achieves comparable performance to DEFO across a wide variety of scenarios. However, there are still another important feature: *the execution cost*, which can be a crucial aspect to assess whether the

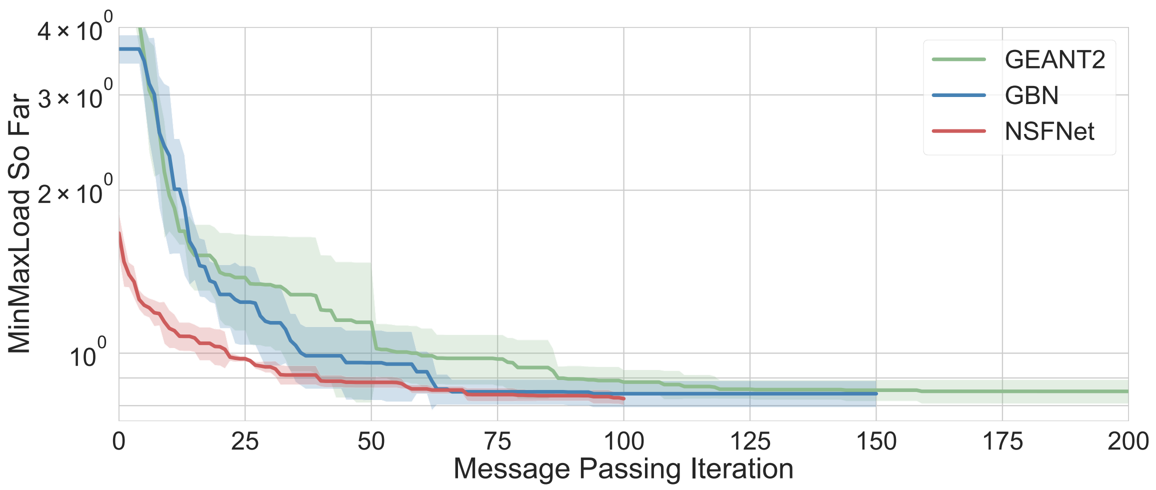


Figure 4.7: Evolution of the MinMax link load *so far* along an episode when applying our model (trained in NSFNet+GEANT2) respectively to NSFNet, GBN, and GEANT2. Plots show the mean and std. dev. over 5 runs, each considering a different TM.

proposed ML-based solution can achieve reasonable execution times for near real-time operation, as in DEFO.

With the above objective in mind, in this section we first analyze the impact of one main hyperparameter of our MARL system, which is the episode length T. This is the maximum number of optimization steps that the MARL system needs to execute before producing a good set of link weights. Given that in our framework only one of the agents selects an action (i.e., increase its link weight) at each time-step of the episode, we expect a straightforward correlation between the number of steps and the total amount of links in the network: the larger the number of links, the larger should potentially be the episode explorations to achieve a good configuration. Finding the exact relation, though, depends on multiple complex factors (e.g., the distribution of links, the initialization of weights, the estimated traffic demands).

By exploring systematically a variable number of steps in the three topologies considered above (NSFNet, GBN, GEANT2), we have empirically found that with an episode length ≈2-3 times the number of links in the network, our system reaches its best performance –which is comparable to the near-optimal results of DEFO, as observed in previous sections. For instance, in our experiments it is sufficient to define T=100 for NSFNet, T=150 for GBN, and T=200 for GEANT2. This can be observed in Figure 4.7, which shows the evolution of the maximum link utilization achieved by our MARL system along an episode in the three network topologies.

4.3.7 Cost Evaluation

Considering the previous evaluation on the episode length (Sec. 4.3.6), in this section we aim to evaluate the execution time of our solution to reach its best optimization potential.

This is probably the main advantage that we can expect from ML-based solutions w.r.t. to near-optimal state-of-the-art TE techniques, such as DEFO. Indeed, if we analyze the main breakthroughs of RL in other fields (e.g., [91]), we can observe they have been mainly achieved in complex online decision-making and automated control problems. Note also that, after training, our multi-agent system is deployed in a distributed way over the network, thus distributing the computation of the global TE optimization process.

Table 4.1 shows the execution time of our MARL+GNN trained system for the three real-world topologies used in our evaluation: NSFNet, GBN and GEANT2. Moreover, we also simulated executions over two synthetic networks –SYNT500 (500 nodes, 1750 links) and SYNT1000 (1000 nodes, 3200 links)– in order to analyze the cost of our distributed system in larger networks. As we can see, the execution time of our solution scales in a very cost-effective way with respect to the size of the network; from the order of milliseconds in NSFNet, GBN and GEANT2, to the tens of seconds in the SYNTH1000 network, with thousands of nodes and links. In contrast, DEFO requires 3 minutes for optimizing networks of several hundreds of nodes [18]. This shows an important reduction in the execution cost of our solution; particularly it represents one order of magnitude improvement in the case of the largest network (SYNTH1000).

We note, though, that this improvement is achieved at the expense of exchanging additional GNN messages between nodes (MPNN). We show in Table 4.1 the MPNN communication cost in terms of the average link overhead resulting from such extra messages. As expected, the cost is quite similar in all topologies, as the messaging overhead of our distributed protocol is directly proportional to the average node degree (i.e., number of neighbors) of the network, and computations are distributed among all nodes. In particular, we can see that the average link overhead only involves a bandwidth of few MB/s per link independently of its capacity, which can reasonably have a negligible impact in today's real-world networks with 10G/40G (or even more) interfaces.

4.4 Extended Evaluation

In our second work about MAGNNETO-TE [33], we deep dived into the question of our first paper by formulating an enhanced MAGNNETO framework (the final version that we described in Chapter 3) and performing a much more comprehensive evaluation. We summarize below the main differences of the extended evaluation presented in this Section with respect to the previous exploratory one:

- Now MAGNNETO formulates the TE problem as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP), which enables to achieve a more functional MARL setting. Instead, the previous version operated over a classical MDP, where agents must take actions sequentially in a synchronized manner.

- Therefore, in the experiments showed below MAGNNETO-TE supports simultaneous actions at each RL optimization step. This dramatically reduces the execution time (up to 10x in our experiments) with respect to the previous framework, which was limited by design to one action per step.
- Lastly, we present in this section an extensive evaluation including 75+ real-world topologies, large-scale scenarios (up to 153 nodes), and a benchmark consisting of a representative collection of advanced TE optimizers. In contrast, the previous evaluation only considered 3 different topologies of limited size (up to 24 nodes), and the results were compared against a single TE solver (DEFO).

We begin by describing the considered baselines as well as the setup used in our evaluations. The rest of the section is devoted to analyze the results.

4.4.1 Baselines

In this section we describe the set of baselines we use to benchmark MAGNNETO-TE in our evaluation. We particularly consider a well-established standard TE mechanism and three advanced TE optimizers.

The first baseline is labeled as "Default OSPF", a simple heuristic widely used in today's ISP networks. In Default OSPF, link weights are inversely proportional to their capacities and traffic is split over multiple paths using ECMP. In our experiments, all performance results are expressed in terms of their improvement with respect to Default OSPF.

As state-of-the-art TE benchmarks, we consider the following set of centralized algorithms provided by REPETITA [73]:

- TabuIGPWO (IGP Weight Optimizer, based on [16]): This algorithm runs a LS to find the OSPF weights that minimize the load of the maximally-utilized link. TabuIGPWO requires more execution time than the rest of baselines, but represents a classical TE optimizer that operates in the same optimization space than MAGNNETO-TE (i.e., OSPF link weight configuration).
- DEFO (Declarative and Expressive Forwarding Optimizer) [18]: It uses CP and SR [75] to optimize routing configurations in the order of minutes. To this end, DEFO reroutes traffic paths through a sequence of middlepoints, spreading their traffic over multiple ECMP paths.
- SRLS (SR and LS) [17]: By leveraging LS and SR, SRLS achieves similar –or even better– performance than DEFO at a lower execution time. It also implements ECMP, and reroutes traffic paths through a sequence of middlepoints.

Particularly, SRLS and DEFO represent state-of-the-art TE optimizers obtaining close-to-optimal performance on several network optimization goals –one of them being our intradomain TE goal of minimizing the most loaded link. To this end, both optimizers leverage

SR, which enables to define overlay paths at a source-destination granularity. In contrast, MAGNNETO-TE and TabuIGPWO operate directly over standard OSPF-based networks with destination-based routing.

4.4.2 Experimental Setup

We compare MAGNNETO against the previously defined TE baselines in all our experimental settings, which involve 82 different real-world topologies: NSFNet, GBN, and GEANT2 from [85], and 79 networks from the Internet Topology Zoo dataset [74]. In this section we provide more low-level technical details of MAGNNETO-TE's configuration, required to reproduce the results.

Regarding the length T of the training and evaluation RL-based episodes, it varies depending on the network topology size and the number of simultaneous actions allowed (more details below in Sec. 4.4.3). At the beginning of each episode, the link weights are randomly selected as an integer in the range $[1, 4]$, so our system is evaluated over a wide variety of scenarios with random routing initializations. From that point on, at each step of an episode one or several agents can modify their weight by increasing it by one unit.

Taking [86] as a reference for defining the hyperparameters' values of the solution, we ran several grid searches to appropriately fine-tune the model. The implemented optimizer is Adam with a learning rate of $3{\cdot}10^{-4}$, β=0.9, and ϵ=0.01. Regarding the PPO setting, the number of epochs for each training episode is set to 3 with batches of 25 samples, the discount factor γ is set to 0.97, and the clipping parameter is 0.2. We implement the Generalized Advantage Estimate (GAE), to estimate the advantage function with λ=0.9. In addition, we multiply the critic loss by a factor of 0.5, and we implement an entropy loss weighted by a factor of 0.001. Finally, links' hidden states h_e are encoded as 16-element vectors, and in each MPNN forward propagation K=4 message passing steps are executed.

For each experiment, we generate two sets of simulated traffic matrices: uniform distribution across source-destination traffic demands, and traffic distributions following a gravity model [87] –which produces realistic Internet traffic patterns. The training process of MAGNNETO-TE highly depends on the topology size; in a machine with a single CPU of 2.20 GHz, it can take from few hours (≈20 nodes) to few days (100+ nodes).

4.4.3 Multiple Actions and Episode Length

As previously mentioned in Chapter 3, there is a relevant hyperparameter that needs to be further addressed: the episode length T of RL-based episodes, which represents the maximum number of optimization steps that MAGNNETO needs to execute before producing a good set of link weights. In this section we provide more details about its definition in terms of the topology size and the number of simultaneous actions.

Let n be such maximum number of simultaneous actions allowed at each time-step t of the episode. When imposing n=1 –i.e., only one link weight changes per time-step–, we have

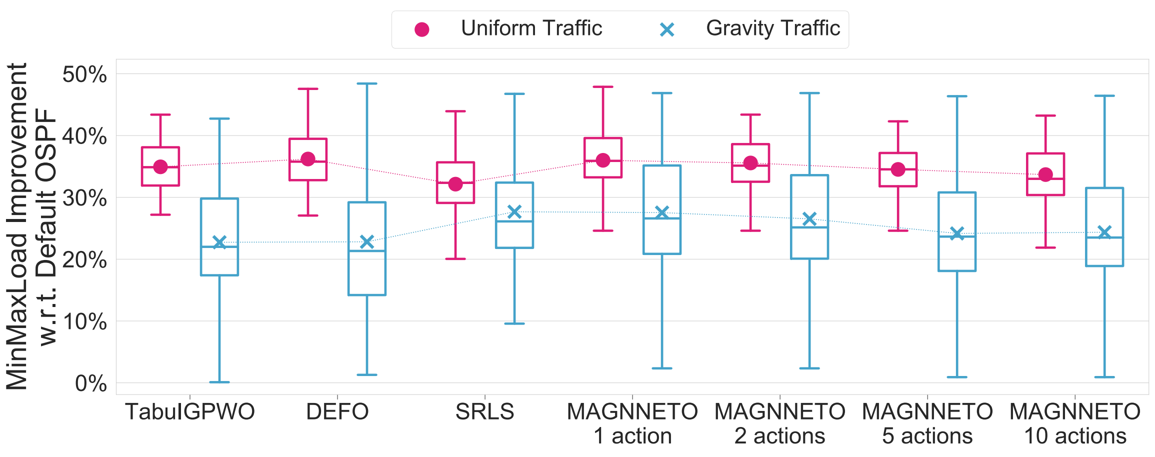


Figure 4.8: Evaluation of MAGNNETO-TE for different number of simultaneous actions $n \in \{1, 2, 5, 10\}$, each of them considering an episode length of $T = 150/n$. The training only considers samples of NSFNet and GEANT2 topologies, and the evaluation is performed over 100 unseen TMs on the GBN topology. Each MAGNNETO model and baseline optimizer is trained and/or evaluated twice for uniform and gravity-based traffic profiles; markers represent the mean of these results, and we also include the corresponding boxplots.

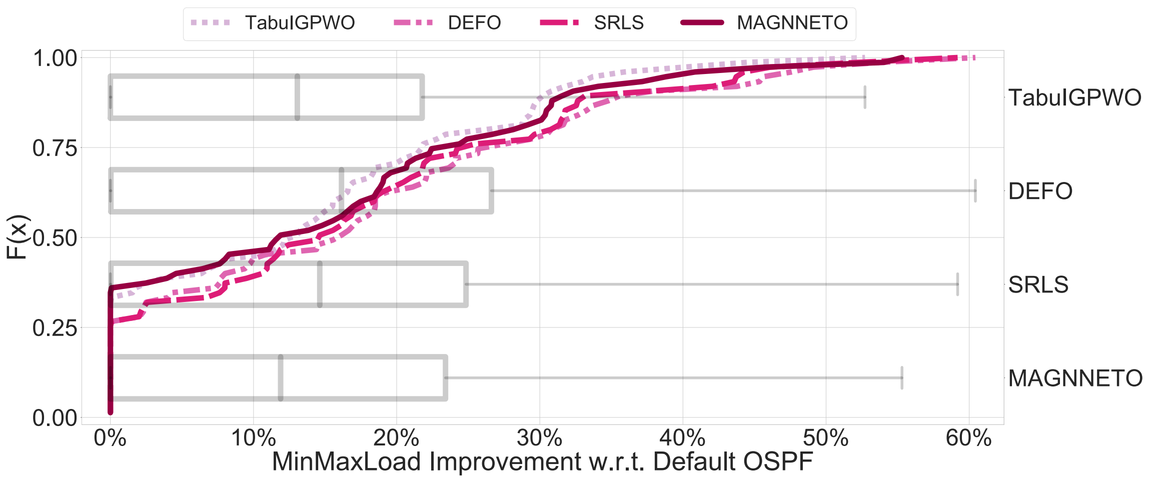
empirically found that MAGNNETO requires an episode length of ≈2−3 times the number of links in the network to reach its best performance. This is in line to what we already observed in previous Section 4.3, based on our preliminary work [92]. However, whereas [92] was subject to n=1 by design, MAGNNETO-TE allows taking n>1 actions at each time-step, which can potentially reduce the number of required optimization steps (i.e., speed up the optimization process).

Figure 4.8 shows that the length T of the episode –which directly relates to the execution time– can be reduced proportionally by n without a noticeable performance loss. In particular, the model with n=10 actually reduces by one order of magnitude the execution time of the 1-action model, but still achieves comparable performance to the state-of-the-art optimizers of our benchmark –for both traffic profiles, and evaluating on a topology not previously seen in training.

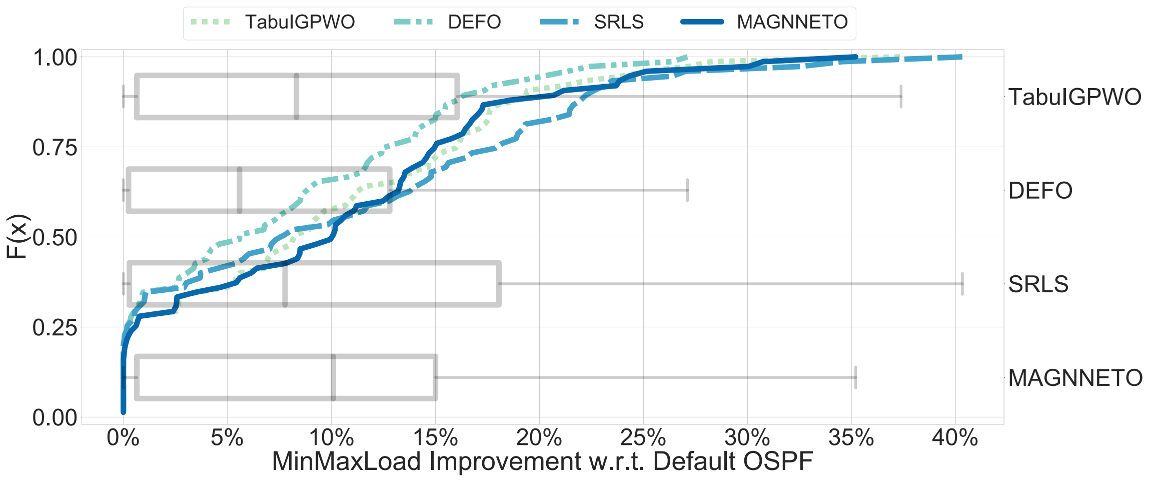
Given the good trade-off that provides allowing more than one action at each time-step, for the rest of our experiments we fine-tuned the number of actions n and the episode length T to balance a competitive performance with the minimum possible execution time. Later in Section 4.4.6 we will analyze in detail the execution cost of MAGNNETO.

4.4.4 Generalization over Unseen Topologies

At the introduction of this chapter we argued the importance of generalization in ML-based solutions, which refers to the capability of the solution to operate successfully in other networks where it was not trained. In this section, we bring MAGNNETO-TE under an intensive evaluation in this regard.



(a) TopologyZoo Uniform



(b) TopologyZoo Gravity

Figure 4.9: Evaluation of MAGNNETO-TE's generalization capability for (a) uniform and (b) gravity traffic. Each point of the CDF corresponds to the mean *MinMaxLoad* improvement over 100 TMs for one of the 75 evaluation topologies from Topology Zoo [74], and boxplots are computed based on these mean improvement values as well. Both the uniform (a) and gravity (b) MAGNNETO-TE models evaluated were trained exclusively on samples from the NSFNet and GEANT2 topologies [85].

In our experiments, MAGNNETO only observes NSFNet (14 nodes, 42 links) and GEANT2 (24 nodes, 74 links) samples during training [85], whereas the evaluation is performed over a subset of 75 networks from the Topology Zoo dataset [74] including topologies ranging from 11 to 30 nodes, and from 30 to 90 links. More in detail:

- We train two MAGNNETO models, one for each traffic profile (uniform and gravity).
- Each model is trained observing 50 different TMs –either uniform or gravity-based, depending on the model– alternating between the NSFNet and GEANT2 topologies.

- Each of these two trained models is evaluated over 100 different TMs –again, either uniform or gravity-based– on each of the 75 topologies from Topology Zoo.

Overall, this experimental setup comprises $7,500$ evaluation runs for each traffic profile, which we summarize in Figures 4.9a and 4.9b, respectively for uniform and gravity-based loads. In particular, note that we first compute the mean *MinMaxLoad* improvement of MAGNNETO-TE –and the baselines– over the 100 TMs of each evaluation network, obtaining a single value for each of the 75 topologies. Thus, in these figures we represent the corresponding CDF and boxplot of the 75-sized vector of mean improvement values for each TE optimizer.

In both traffic scenarios MAGNNETO-TE achieves comparable performance to the corresponding best performing benchmark –DEFO when considering uniform traffic and SRLS for gravity. In fact, MAGNNETO-TE outperforms TabuIGPWO, improves DEFO with gravity-based traffic, and lies within a 2% average improvement difference with respect to SRLS in both cases. We reckon that these represent remarkable results on generalization; as far as we know, this is the first time that a ML-based model consistently obtains close performance to state-of-the-art TE optimizers on such a large and diverse set of topologies not previously seen in training.

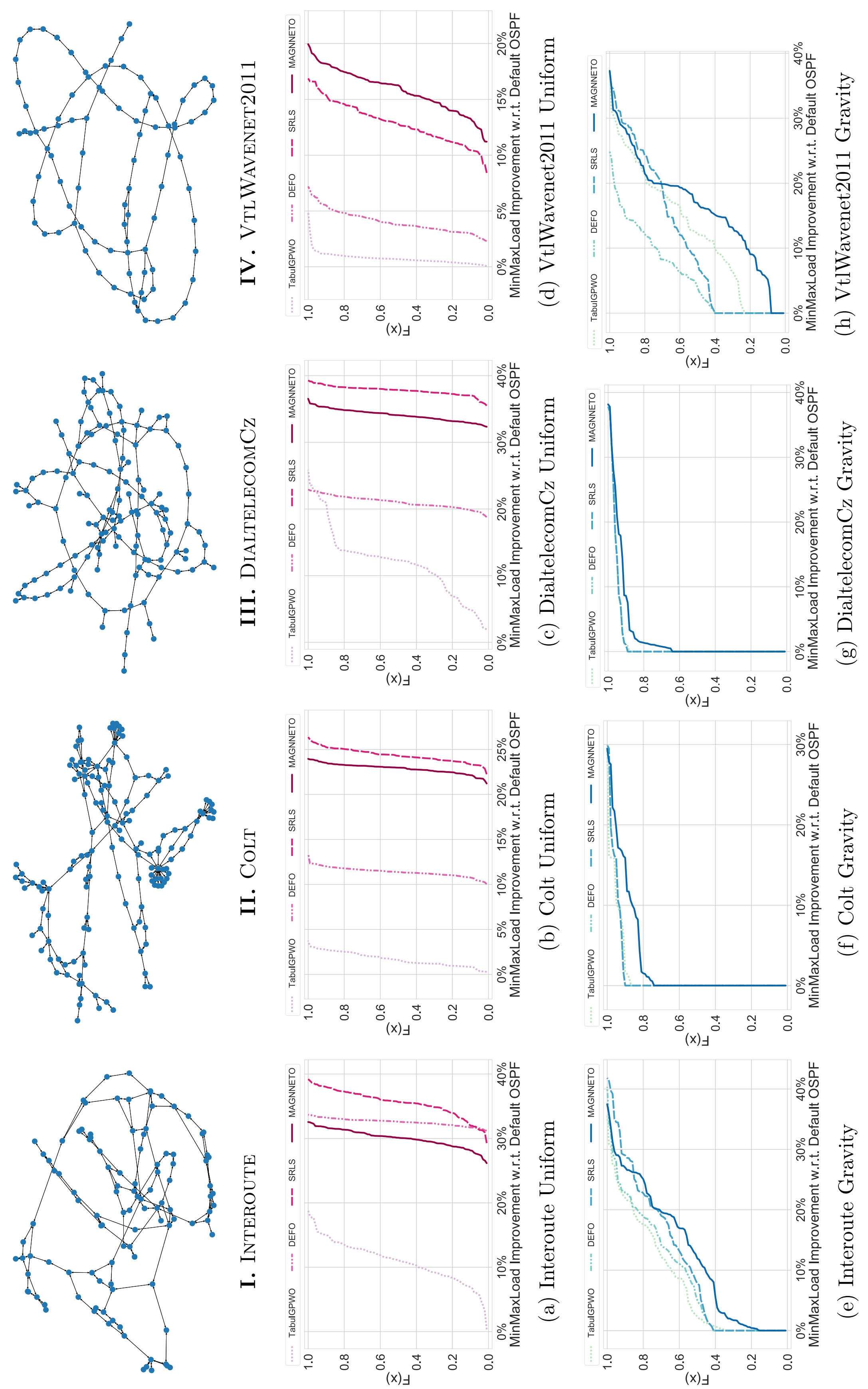


Figure 4.10: Evaluation of MAGNNETO-TE on traffic changes in four large real-world topologies (I-IV) from the Topology Zoo dataset [74], both for uniform ((a)-(d)) and gravity-based ((e)-(h)) traffic loads. A MAGNNETO-TE model is trained for each network and traffic profile, and then evaluated on the same topology over 100 unseen TMs. CDFs represent the *MinMaxLoad* improvement results of each optimizer for those 100 evaluation TMs.

4.4.5 Traffic Changes in Large Topologies

After evaluating the generalization capabilities of MAGNNETO-TE, we aim to test the performance of our method over traffic changes in large networks, where the combinatorial of the optimization process might dramatically increase. Having considered networks up to 30 nodes and 90 links so far, for this set of experiments we arbitrarily select four large real-world topologies from Topology Zoo [74]: Interoute (110 nodes, 294 links), Colt (153 nodes, 354 links), DialtelecomCz (138 links, 302 links) and VtlWavenet2011 (92 nodes, 192 links). Figures 4.10.I-IV depict these topologies.

In these experiments, for each traffic profile (uniform or gravity) we train a MAGNNETO-TE model on each network. Then, we evaluate models on the same topology where they were trained, over 100 different TMs not previously seen in training. Figures 4.10.a-d and 4.10.e-f show the corresponding CDF of all these evaluations, considering uniform and gravity traffic loads respectively.

As we can see, with uniform traffic SRLS is clearly the best performing baseline, achieving a remarkable overall improvement gap with respect to the other two benchmarked optimizers. However, in this scenario MAGNNETO-TE is able to obtain similar improvements to SRLS, slightly outperforming it in VtlWavenet2011. On the other hand, results with gravity-based traffic suggest that Default OSPF already provides with low-congested routing configurations in scale-free networks when considering more realistic traffic. Despite this fact, MAGNNETO-TE turns out to be the overall winner in the comparison with gravity loads, consistently achieving lower congestion ratios for a large number of TMs in all four topologies.

In short, in all scenarios MAGNNETO-TE attained equivalent –or even better– performance than the advanced TE optimizers benchmarked. These results evince its potential to successfully operate in large computer networks.

4.4.6 Execution Cost

Lastly, in this section we evaluate the execution cost of MAGNNETO-TE. In particular, we measure the impact of the message communications involved when running our distributed solution, as well as compare its execution time against the considered set of state-of-the-art TE baselines; Table 4.2 gathers these results for several variable-sized networks used in the previous evaluations.

Taking into account the recommendations of REPETITA [17], as well as analyzing the results provided in the original works [16–18], we defined the following running times for each of our benchmarks: 10 minutes for TabuIGPWO, 3 minutes for DEFO, and 1 minute for SRLS.

At first glance, the execution time of MAGNNETO-TE becomes immediately its most remarkable feature. Particularly, it is able to obtain subsecond times even for the larger network of our evaluation (Colt). Indeed, as previously discussed in Section 4.4.3 these times could be further reduced by allowing multiple simultaneous actions. For instance, by considering up to 10 simultaneous actions, MAGNNETO-TE can run 3 orders of magnitude faster than the most rapid state-of-the-art TE optimizer. This relevant difference can be explained by the fact that MAGNNETO-TE's distributed execution naturally parallelizes the global optimization process across all network devices (i.e., routers); in contrast, typical TE optimizers rely on centralized frameworks that cannot benefit from this.

Such decentralization comes at the expense of the extra message overhead generated by the MPNN. In this context, Table 4.2 shows that the link overhead produced by MAGNNETO-TE (few MB/s) can reasonably have a negligible impact in today's real-world networks with 10G/40G (or even more) interfaces. Moreover, note that this cost is quite similar in all topologies; this is as expected, given that the messaging overhead of the GNN-based communications is directly proportional to the average node degree of the network, and computations are distributed among all nodes.

To sum up, our results show that MAGNNETO-TE is able to attain equivalent performance to state-of-the-art centralized TE optimizers –even in topologies not previously seen in training– with significantly lower execution time, and with an affordable message communication overhead.

	NSFNet	**GBN**	**GEANT2**	**VtlWavenet2011**	**Interoute**	**DialtelecomCz**	**Colt**
(#nodes, #links)	(14,42)	(17,52)	(24,74)	(92,192)	(110,294)	(138,302)	(153,354)
Link Overhead* (MB/s)	1.20	1.32	1.20	0.83	1.28	0.91	1.01
Execution Time (s)							
TabuIGPWO [16]	600	600	600	600	600	600	600
DEFO [18]	180	180	180	180	180	180	180
SRLS [17]	60	60	60	60	60	60	60
MAGNNETO-TE [n actions]	$0.08/n$	$0.12/n$	$0.16/n$	$0.42/n$	$0.64/n$	$0.66/n$	$0.78/n$

*Average value, with an extra 20% message size for headers and metadata.

Table 4.2: MAGNNETO-TE Execution Cost: Average link overhead and execution time –in terms of the maximum number of simultaneous actions allowed– for variable-sized network topologies.

4.5 Related Work

Recently, numerous solutions based on RL have been proposed to solve complex networking problems, especially in the context of routing optimization and TE [70, 72, 93]. However, current state-of-the-art RL-based TE solutions fail to generalize to unseen scenarios (e.g., different network topologies) as the implemented traditional neural networks (e.g., fully connected, convolutional) are not well-suited to learn and generalize over data that is inherently structured as graphs. In [71], the authors design a RL-based architecture that obtains better results than Shortest Path and Load Balancing routing. Regarding MARL-based solutions [22, 94], most of them suffer from the same lack of topology generalization. An exception to that is the work of [23], an interesting MARL approach for multi-region TE that consistently outperforms ECMP in several scenarios, although it is not benchmarked against state-of-the-art TE optimizers.

GNNs [26, 95], and in particular MPNN [45], precisely emerged as specialized methods for dealing with graph-structured data; for the first time, there was an AI-based technology able to provide with topology-aware systems. In fact, GNNs have recently attracted a large interest in the field of computer networks for addressing the aforementioned generalization limitations. The work from [85] proposes to use GNN to predict network metrics and a traditional optimizer to find the routing that minimizes some of these metrics (e.g., average delay). Authors of [96] propose a novel architecture for routing optimization in Optical Transport Networks that embeds a GNN into a centralized, single-agent RL setting that is compared against Load Balancing routing.

Narrowing down the use case to intradomain TE, we highlight the work of [97], whose premise is similar to ours: the generation of easily-scalable, automated distributed protocols. For doing so, the authors also use a GNN, but in contrast to our approach they are focused on learning routing strategies that directly imitate already existing ones –shortest path and min-max routing– and compare their solution against these ones. This is the reason why they did not implement a RL-based approach, but instead a semi-supervised learning algorithm, therefore guiding the learning process with explicit labeled data. In fact, so far the very few works that combine GNNs with a MARL framework [27, 28] are theoretical papers from the ML community, and none of them apply to the field of networking.

4.6 Discussion

Intradomain Traffic engineering (TE) is nowadays among the most common network operation tasks, and has a major impact on the performance of today's ISP networks. As such, it has been largely studied, and there are already some well-established TE optimizers that deliver near-optimal performance in large-scale networks. During the last few years, state-of-the-art TE solutions have systematically competed for reducing execution times (e.g., DEFO [18], SRLS [17]), thus scaling better to carrier-grade networks and achieving faster reaction to traffic changes. In this context, ML has attracted interest as a suitable technology for achieving faster execution of TE tasks and –as a result– during recent years the networking community has devoted large efforts to develop effective ML-based TE solutions [23, 70–72]. However, at the time of this writing no ML-based solution had shown to outperform state-of-the-art TE optimizers.

In this chapter we have presented MAGNNETO-TE, a novel ML-based framework for intradomain TE optimization. Our system implements a novel distributed architecture based on MARL andGNN. In our evaluation, we have compared MAGNNETO-TE with a set of non-ML-based TE optimizers that represent the state of the art in this domain. After applying our system to 75+ real-world topologies, we have observed that it achieves comparable performance to the reference TE benchmarks. However, MAGNNETO-TE offers considerably faster operation than these state-of-the-art TE solutions, reducing execution times from several minutes to sub-second timescales in networks of 100+ nodes. In this context, MAGNNETO-TE was especially designed to perform several actions at each RL optimization step, which enables to considerably accelerate the optimization process. Particularly, we have seen that our system was able to perform up to 10 actions in parallel with no noticeable decrease in performance. These results lay the foundations for a new generation of ML-based systems that can offer the near-optimal performance of traditional TE techniques while reacting much faster to traffic changes.

Last but not least, we have shown that the proposed system offers strong generalization power over networks unseen during the training phase, which is an important characteristic from the perspective of deployability and commercialization. Particularly, generalization enables to train ML-based products in controlled testbeds, and then deploy them in different real-world networks in production. However, this property has been barely addressed by prior ML-based TE solutions. In contrast, MAGNNETO-TE has demonstrated to generalize succesfully over a wide and varied set of 75 real-world topologies unseen during training. The main reason behind this generalization capability is that the proposed system implements internally a GNN that structures and processes network information as graphs, and computes the information on distributed agents that communicate with their neighbors according to the underlying graph structure (i.e., the network topology).

Chapter 5

MAGNNETO-CC: Congestion Control in Datacenters

The last decade has witnessed an ever-growing interest on optimizing Datacenter Networks (DCNs) given the critical services they run and the high capital and operational expenses these infrastructures entail. Among the large spectrum of traffic optimization mechanisms [98], Congestion Control (CC) has been especially explored in the literature, with a large corpus of outstanding proposals aiming at accurately control traffic in high-speed DCNs [24, 99–105]. Nowadays, most production DCNs run two main CC protocols: (*i*) Datacenter Transmission Control Protocol (DCTCP) [99], for datacenters based on the traditional Transmission Control Protocol/Internet Protocol (TCP/IP) stack, and (*ii*) Datacenter Quantized Congestion Notification (DCQCN) [100], for emerging Remote Direct Memory Access (RDMA) based DCNs. Both CC schemes rely on congestion notifications raised by intermediate switches in the network — known as the Explicit Congestion Notification (ECN) mechanism [106]. This is the only feedback that end-hosts receive to dynamically adapt the flow rate. The ECN configuration thus becomes a crucial aspect for optimizing traffic in today's datacenters [24, 107], and finding the optimal ECN parameters is a complex and time consuming task. Nowadays, network experts struggle to set good ECN configurations after a thorough characterization of the traffic workload, and under the assumption that the network topology and traffic are sufficiently static. As a result, they end up selecting configurations that can work well on average, while being conservative enough to absorb traffic microbursts and avoid queue buildup [107].

However, traffic in modern high-speed DCNs is more and more dynamic. For example, in emerging applications —such as distributed cloud storage or Machine Learning (ML)— it is very frequent to find incast events (synchronous many-to-one connections), which put great pressure over specific switch ports for short time spans. At the same time, production DCNs experience failures frequently [108, 109]. This means topology changes that break a main design principle of today's DCNs: network symmetry. Failures cause network imbalance, which may lead to severe performance degradation (up to 40% throughput reduction in real-world networks [110]).

In this vein, ML has raised a special interest as a suitable technique to dynamically optimize CC in DCNs. Nowadays, we can attest some pioneering ML-based CC proposals, such as AuTO [111], Aurora [112], or Orca [113]. However, these solutions are not compatible with widely deployed equipment in datacenters, as they propose to re-implement the network stack. A more recent solution, ACC [24], proposes to perform in-network optimization by dynamically adapting the ECN configuration on switches. This solution has shown outstanding performance in production environments and it is compatible with current datacenter equipment running widely deployed ECN-based CC protocols (e.g., DCTCP, DCQCN). Nevertheless, ACC is designed for online training; i.e. it gradually learns how to adapt to the current network conditions. As a result, it may suffer from critical transient performance degradation when traffic changes. In general, online training is not always appropriate in production environments, as: *(i)* it carries an implicit uncertainty on what would be the resulting performance of agents after re-training, *(ii)* the training adds an extra execution cost, and *(iii)* it may be not compatible with legacy hardware —or simply training takes too much time due to the computational requirements for training models there.

This chapter presents MAGNNETO-CC, the adaptation of the general MAGNNETO framework introduced in Chapter 3 for in-network CC optimization. In particular, it does not need further training once deployed; by design, our solution is able to adapt to varying DCN traffic conditions despite it being trained in a fully offline manner. Similar to ACC [24], MAGNNETO-CC dynamically optimizes the ECN configuration on switches, and it is compatible with widely deployed CC protocols (e.g., DCTCP, DCQCN). However, our method is based on a novel combination of Multi-Agent Reinforcement Learning (MARL) and Graph Neural Networks (GNNs) that, after training, produces a single agent implementation that can be deployed in a distributed way on switches to optimize the ECN configuration at the interface level. In contrast to previous proposals, deployed MAGNNETO agents do communicate with adjacent agents to get local context, and they actually learn how to cooperate to optimize the global Flow Completion Time (FCT); we argue this local information and cooperation can potentially help agents to take better ECN adjustments.

Remark: Analogously to what we did in previous Chapter 4, in this chapter we will also interchangeably employ the acronyms MAGNNETO-CC and MAGNNETO to denote the adaptation of the framework, in this case to CC. Regarding the general method introduced in Chapter 3, we shall consistently address it as the general MAGNNETO framework as well.

We evaluate MAGNNETO-CC under a diverse spectrum of scenarios with DCQCN, including different real-world traffic workloads unseen during the training phase. Also, we test how this solution behaves under aggressive incast events and drastic topology changes, such as link failures or network upgrades. We compare the performance with respect to: (*i*) a static ECN setting used in Alibaba's production networks [103], and (*ii*) ACC [24], the previously mentioned state-of-the-art ML-based solution for dynamic ECN tuning. In our evaluation, we use the average FCT slowdown as a reference, which is considered a fairness metric that normalizes FCT with respect to flow size [114–116]. Our experimental results show that MAGNNETO-CC achieves improvements of up to 20% in the average FCT slowdown with respect to ACC without re-training. Likewise, by tuning the ECN configuration MAGNNETO-CC learns to optimize flow-level performance while keeping short queue lengths (reduction of 38.0-85.7% with respect to ACC). This may be beneficial to achieve stability under unpredictable traffic microbursts [107], and it is a trend already seen in other near-optimal state-of-the-art CC mechanisms relying on advanced telemetry, such as HPCC [103].

The remainder of this chapter is as follows. Section 5.1 describes in detail the CC scenario, and we summarize the main challenges of this use case in Section 5.2. Next, Section 5.3 describes how we adapt the general MAGNNETO framework to perform CC optimization in datacenters. In Section 5.4 we show our extensive MAGNNETO-CC evaluation. Section 5.5 summarizes the main existing works related to this application, and lastly Section 5.6 concludes the chapter with a discussion of the results.

5.1 Network Scenario

CC has been extensively studied in the past. As a result, there exists a plethora of pioneering solutions for DCNs tackling the problem from different angles, such as RTT-based [101,104], credit-based [102,117], or telemetry-based [103,118] mechanisms. Nowadays most production DCNs implement two main well-established CC protocols: DCTCP [99], and DCQCN [100]. The former is the main standard in traditional networks based on the TCP/IP stack, while the latter is the *de facto* standard in modern RDMA-based networks[1]. Both state-of-the-art mechanisms — as well as their enhanced schemes [119, 120] — rely on ECN [106], so that switches mark packets when they experience congestion, and end-hosts dynamically adapt their transmission rate accordingly to this congestion feedback. More advanced CC mechanisms, such as HPCC [103] or Swift [104], have been proposed and validated in production networks, showing outstanding performance compared to their contemporaries. However, these solutions require features that are not widely supported by legacy datacenter equipment, such as in-band network telemetry or accurate real-time RTT measurements [118, 121, 122].

This part of the dissertation focuses on in-network optimization of widely deployed ECN-based CC protocols (e.g., DCTCP, DCQCN). MAGNNETO-CC attempts to optimize

[1]RDMA is a link-level technology that optimizes memory access across distributed nodes in DCNs.

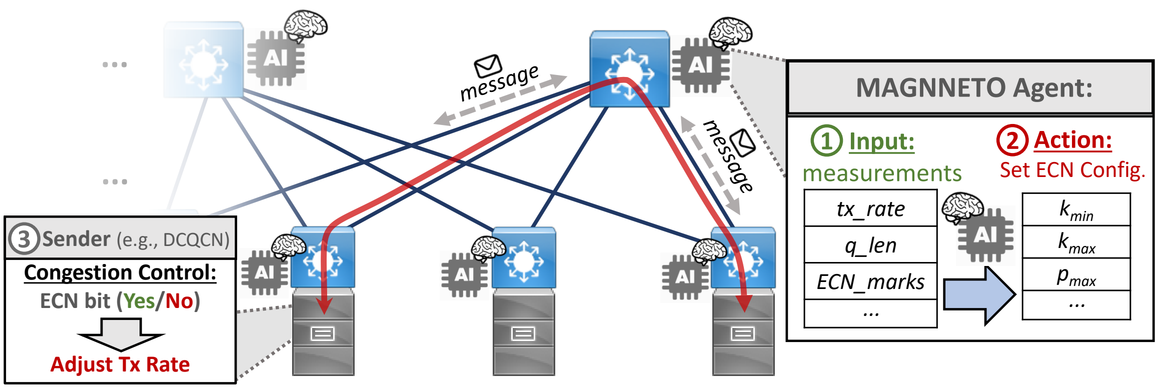


Figure 5.1: Schematic representation of MAGNNETO-CC: Distributed in-network CC optimization.

the handling of congestion notifications in switches, which is a crucial component of CC protocols to efficiently optimize traffic [24]. In this context, both DCTCP and DCQCN implement a similar approach: switches mark the Congestion Experienced (CE) bit of packets in case the queue length exceeds some predefined thresholds. DCTCP adopts a hard cutoff, i.e., all packets are marked when the queue length exceeds a certain value k. Instead, DCQCN implements a softer RED-like probabilistic approach based on three ECN configuration parameters $\{k_{min}, k_{max}, p_{max}\}$ [123].

$$p_{mark}(q_{len}) = \begin{cases} 0 & \forall\, q_{len} \in [0, k_{min}) \\ \frac{q_{len}-k_{min}}{k_{max}-k_{min}} \cdot p_{max} & \forall\, q_{len} \in [k_{min}, k_{max}]\,, \\ 1 & \forall\, q_{len} \in (k_{max}, \infty) \end{cases} \tag{5.1.1}$$

where q_{len} is the instantaneous queue length in the NIC. These parameters have a significant impact on the resulting network performance (e.g., FCTs), and their optimal values are highly dependent on the current traffic conditions [24]. This poses a great challenge on how to dynamically adapt these values to better exploit network resources, as further discussed in Section 5.2.

In this context, MAGNNETO-CC deploys a set of distributed agents in switches that communicate between them to jointly optimize the ECN configuration on NICs. The operational workflow —Figure 5.1 shows a schematic representation— is as follows:

1) Measurement collection on NICs (Fig. 5.1; step ①): First, the distributed agents of MAGNNETO-CC retrieve basic measurements from their local NICs. Specifically, they collect the bytes transmitted by the NIC (tx_rate), the queue length (q_len), and the number of packets marked by ECN in the past (ECN_marks). These measurements are commonly supported by commercial switches [124–126], and can be locally obtained with low computational overhead at microsecond timescales [24].

2) MAGNNETO-CC optimize the ECN configuration (Fig. 5.1; step ②): Once agents collect NIC measurements, they start a communication with other agents deployed in adjacent switches to gain a local context. This communication is done through a novel NN-driven message passing, where agents exchange messages directly encoded by NN modules in order to find the best ECN settings in their associated NICs. This communication only requires to exchange few bytes with neighboring switches and can take few μs (the base link propagation delay in production DCNs is typically 1-2μs [24, 103]). To set the new ECN parameters — e.g., $\{k_{min}, k_{max}, p_{max}\}$ in DCQCN — agents can directly interface with forwarding chips using their API, which is typically vendor-specific. At this point, switches start to mark packets according to the new ECN settings. More details about the architecture, inner workings, communication overhead and performance of MAGNNETO-CC are described in Sections 5.3 and 5.4.

3) End-hosts adjust the flow rate (Fig. 5.1; step ③): Lastly, the CC protocol executed at end-hosts (e.g., DCTCP, DCQCN) adjusts the flow transmission rate based on the ECN feedback. The process is as follows: if a host receives an ECN-marked packet, it notifies it to the sender in the corresponding ACK. When the sender receives the ACK, it re-computes the flow rate according to the protocol-specific algorithm (e.g., Additive-Increase/Multiplicative-Decrease). This CC mechanism thus enables to gradually react at one-RTT timescales ($\approx$10μs in high-speed DCNs [24, 103, 104]). Note that the in-network optimization mechanism of MAGNNETO-CC is orthogonal and complementary to the selection of the flow rate control algorithm (e.g., DCTCP, DCQCN). MAGNNETO-CC is compatible with any ECN-based CC protocol and can be deployed along with any other well-established traffic optimization techniques, such as flow scheduling [114, 115].

MAGNNETO-CC follows a practical approach to dynamically optimize the ECN setting in switches, which indirectly affects packet marking and the flow rate adaptation at end-hosts. Note that more accurate control could be achieved by marking traffic at finer granularity levels (e.g., packet, flow-level). However, this would not be feasible from a practical standpoint given the high-speed traffic volumes in today's DCNs. Instead, MAGNNETO operates at tractable timescales — at the scale of RTTs — and is able to efficiently optimize flow-level performance in DCNs, as shown later in the evaluation of Section 5.4.

5.2 Main Challenges in CC Optimization

This section discusses some key aspects to consider when optimizing CC protocols in high-speed DCNs, which have driven the design of MAGNNETO-CC.

5.2.1 Performance Tradeoffs in Congestion Control

Nowadays, storage and computation speeds are some orders of magnitude faster than networking operations [104]. Hence, the network becomes the main performance bottleneck in today's datacenters. The main operational goal in DCNs is to maximize the throughput, while keeping low latency at the flow level. At the same time, current DCNs carry heavy-tailed traffic distributions, where a large amount of flows are short and time-sensitive, and a small portion of flows are long and throughput-sensitive [99, 100, 127]. We discuss below the main tradeoffs to consider when optimizing CC in DCNs.

Thoughput vs. Latency

A main challenge when operating DCNs is to keep a good compromise between the throughput for long flows and the latency for short flows. The main logic behind ECN-based schemes is to estimate network congestion based on the queue occupation on NICs; particularly, these mechanisms mark packets on switches according to some predefined thresholds on the queue length, as previously described in Section 5.1. Small queue length thresholds (e.g., $[k_{min},\ k_{max}]$ in DCQCN) lead to aggressively dropping packets and keeping short latency on queues, while large thresholds lead to higher bandwidth utilization at the expense of increased latency. In this context, the FCT metric has been widely accepted as the main performance indicator in datacenters nowadays [128], as it unifies throughput and latency in a single metric. In particular, the *FCT slowdown* metric —which computes the ratio between the actual FCTs and the baseline FCTs if flows were sent at line-rate— further introduces fairness across flows, and it is used in most state-of-the-art works to quantify the performance impact on applications [114–116]. Finally, we note that applications in modern DCNs are mainly based on partition/aggregate design patterns (e.g., web search, advertisement selection, Machine Learning) [99, 103], where jobs are broken into small tasks and farmed out across servers, so then partial results are aggregated to produce the final output. This poses a special interest in minimizing the tail latency experienced by flows (e.g., 95/99-pct of FCT slowdown), as it often dominates the overall application performance [99]. Based on this, in the design and evaluation of MAGNNETO-CC we consider the 95-pct and 99-pct of the FCT slowdown as central performance metrics.

Throughput vs. Stability

Nowadays, DCNs are exposed to highly dynamic traffic patterns, such as incast events, where a large number of servers send traffic to a specific host. These patterns are very frequent in modern applications based on the partition/aggregate principle, as in every aggregation phase distributed workers send partial results in a synchronized way to aggregator nodes. Thus, beyond the canonical optimization goals of DCNs (i.e., throughput and latency), CC mechanisms need to account for stability. That is, to be prepared for rapid traffic variations (e.g., incast events, workload changes). Hence, a new tradeoff arises: conservative ECN settings often underutilize the network (i.e., less thoughput) while avoid fast queue buildup during transient incast events; on the contrary, maximizing network utilization leaves scarce headroom in buffers to absorb traffic microbursts and may lead to severe performance degradation [107]. In MAGNNETO-CC, we introduce this tradeoff by explicitly including the queue length and the throughput (tx_{rate}) in the agents' reward function.

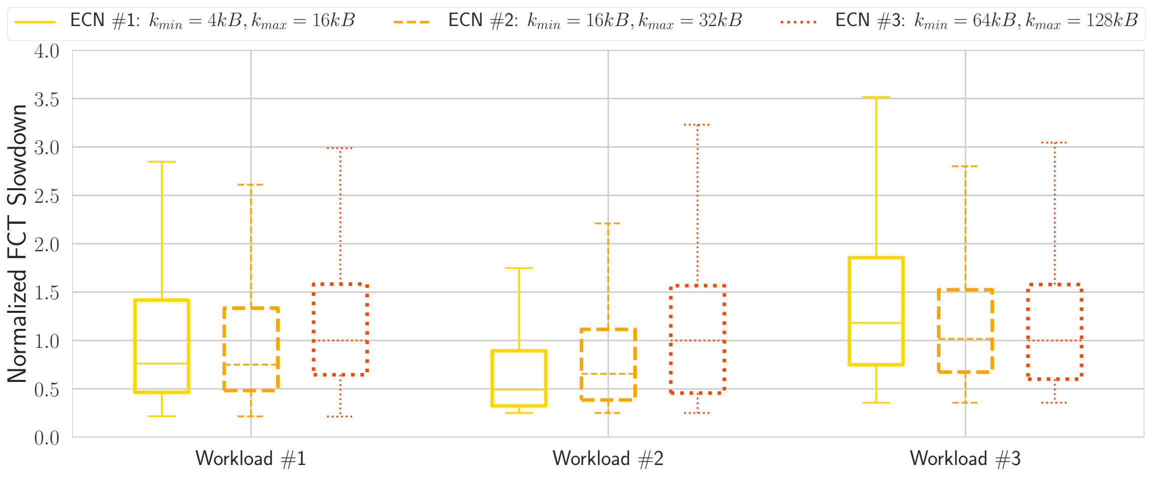


Figure 5.2: Normalized FCTs under various static ECN settings.

5.2.2 ECN Parameters are Hard to Optimize

ECN settings have a large impact on network performance, and they are highly sensitive to traffic conditions [24]. As an example, Figure 5.2 shows some experimental results on a small 2-layer Clos network under three different public real-world traffic workloads (#1: *FB_Hadoop* [127], #2: *WebSearch* [99], #3: *AliStorage* [103]), considering in each of them three different DCQCN configurations; for clarity, values are normalized by the median FCT slowdown of configuration ECN #3. As we can see, FCT values vary considerably depending on the workload. For example, in the case of Workload #1, ECN #1 and ECN #2 far outperform ECN #3 ($\approx$ 25% better on median), while in Workload #3, ECN #3 outperforms the two other configurations.

As a result of the multiple optimization tradeoffs described in the previous subsection and the sensitivity of ECN settings, finding the optimal operational point in DCNs is a cumbersome task. Nowadays, network operators struggle to find good ECN parameters that can perform well on average, while at the same time guaranteeing stability under drastic events (e.g., incast, failures). This often leads to quite conservative ECN settings. The process to refine ECN parameters can typically take weeks to months [24], as network administrators need to make accurate workload characterizations and carefully evaluate alternative configurations under a broad casuistry (e.g., stress testing, check traffic variations, failure scenarios). More importantly, traffic workloads change drastically along the day [24] and networks experience large daily variations on RTT (beyond 3x in real networks [119]). Also, link failures occur very frequently [108, 129], which generate network asymmetries that are highly disruptive for DCNs [109, 110]. For example, the study in [110] states that failures may cause up to 40% throughput reduction despite the typical link redundancy.

5.2.3 Current ML-based Solutions

All this motivates the need for more advanced techniques that can dynamically and timely adapt the ECN configuration to the fast traffic dynamics of nowadays DCNs, and in this context ML becomes a promising mechanism. As a mere example, by taking simple measurements of queue length on ports, we could deploy a system that can predict potential queue buildups due to ongoing traffic peaks and preemptively set more conservative ECN configurations that mitigate performance degradations. Indeed, some pioneering ML-based solutions have already been proposed for traffic optimization in DCNs [24,111–113], showing that ML can be a game-changer in the DCN arena. Motivated by the outstanding results produced by this kind of techniques in the networking field [61], ML seems a promising mechanism to efficiently optimize CC, by dynamically adjusting the ECN configuration. In this context, MAGNNETO-CC stems from the motivation to provide a practical solution for current real-world DCNs, by aligning with the features supported by legacy switches. As a result, MAGNNETO relies on optimizing the configuration of widely deployed ECN-based CC protocols. At the same time, this makes it feasible to operate at tractable timescales that can be gracefully adapted depending on the resources available in the network (e.g., from tens of microseconds to few milliseconds). Also, it implements a GNN architecture where adjacent nodes exchange messages automatically generated by NN modules in order to better cooperate to optimize the overall network performance. Section 5.5 revisits some pioneering ML-based solutions for traffic optimization in DCNs, and further discusses their current limitations (as well as their differences w.r.t MAGNNETO-CC).

5.2.4 Dealing with Decreasing Buffer Size

Last, but not least, there is a fundamental aspect that DCNs are increasingly witnessing as new generations of forwarding devices come to the market. During the last decade, switch capacity and link speeds have grown dramatically ($\approx \times 10$ in six years [130]). However, buffer size on switches have not been scaled at the same pace. This is mainly due to the high cost and technological barriers to scale switch memories to the ever-increasing link speeds. As a result, the ratio buffer size vs. switch capacity has been considerably decreased in the last years. As an example, high-end datacenter switches from Broadcom have reduced this ratio by approximately a factor of 2 in six years (from 2012 to 2018) [130]. This means that buffers now fill up faster, and this trend is expected to continue exacerbating in future datacenter generations. This poses the need for efficient CC mechanisms that can quickly react to the fast traffic dynamics (e.g., incasts). Also, keeping reduced queue length may be more important than ever to achieve stability, e.g., to have sufficient headroom to absorb traffic microbursts. In this vein, MAGNNETO-CC intrinsically accounts for the minimization of the buffer occupancy (i.e., queue lengths) during the optimization process.

5.3 MAGNNETO-CC

The general MAGNNETO framework was carefully adapted to tackle the challenges previously exposed in Section 5.2. This section describes the design and implementation of the resulting MAGNNETO-CC. We first fully contextualize its application to the considered CC optimization scenario (see Section 5.1). Lastly, we describe some details about the deployment of this solution.

5.3.1 General Setting

A DCN can be described in terms of its hardware devices $\mathcal{W} = \{\mathcal{W}_s, \mathcal{W}_h\}$ —where $\mathcal{W}_s$ and $\mathcal{W}_h$ denote the sets of switches and hosts, respectively— and the link connections between them, $\mathcal{L} = \{l = (w^{src}, w^{dst}), w^{src} \in \mathcal{W}$ and $w^{dst} \in \mathcal{W}$ connected$\}$. Therefore, in our graph-based model $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{V})$ of a DCN, we can directly relate the set of nodes $\mathcal{N}$ with hardware devices $\mathcal{W}$, and the set of edges $\mathcal{E}$ with the actual set of network interfaces $\mathcal{L}$. MAGNNETO-CC identifies each egress port of a switch as an agent, i.e. $\mathcal{V} = \{(n^{src}, n^{dst}) \in \mathcal{E}, n^{src} \in \mathcal{W}_s\}$, which in particular allows to define the ECN marking thresholds at a link granularity (i.e., at the interface level)[2]. In this context, we can differentiate two different neighborhoods for each link-based agent $v = (n_v^{src}, n_v^{dst}) \in \mathcal{V}$:

- Ingress neighborhood $\mathcal{B}_i(v)$, defined as the set of links $e = (n_e^{src}, n_e^{dst}) \in \mathcal{E}$ that can potentially inject traffic into v, $\mathcal{B}_i(v) = \{e \in \mathcal{E} | n_e^{dst} = n_v^{src}\}$.
- Egress neighborhood $\mathcal{B}_e(v)$, consisting in the links that can potentially receive traffic from v, $\mathcal{B}_e(v) = \{e \in \mathcal{E} | n_e^{src} = n_v^{dst}\}$.

In addition to that, we implement Q-learning in this case; as described below in next Section 5.3.2, in this CC use case MAGNNETO has to deal with a large state and action spaces, and in our experiments DQN outperformed Actor-Critic methods such as PPO. Therefore, the global function f_θ of the framework (see Section 3.2) becomes the state-action value function Q_θ. Consequently, for each agent $v \in \mathcal{V}$, the readout function takes the final hidden state h_v^K as input, and produces the q-value estimates $Q_\theta(s_v^t, a_v)$ for every possible action $a_v \in \mathcal{A}_v$. Finally, following the standard procedure of Q-learning algorithms, at that time-step t each agent selects the action with the maximum associated q-value, $a_v^t = \arg\max_{a_v} Q_\theta(s_v^t, a_v)$.

5.3.2 Adapting MAGNNETO to CC

This section fully describes how we adapted the operation of MAGNNETO for CC in DCNs (Figure 5.3 shows a schematic operational workflow). We assume that our solution interacts with the environment (i.e. the network) every time interval Δt, which is pre-defined. This enables to make the time evolution discrete, and hence facilitates the episodic

[2]Note that, in practice, port-based agents are deployed and run in their adjacent switches.

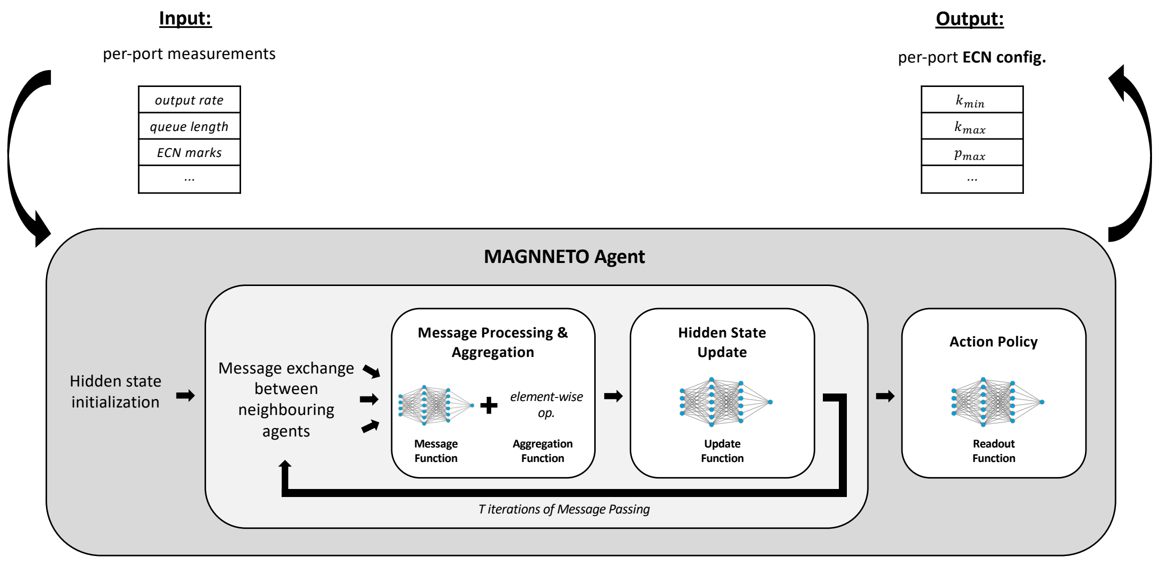


Figure 5.3: Operational workflow of a MAGNNETO-CC agent.

formulation of RL. At each step t of an episode, each agent $v \in \mathcal{V}$ first gathers three relevant NIC-level metrics available at the switch: (i) the port utilization u_v^t (computed as the tx_{rate} normalized by link capacity), (ii) the instantaneous queue length q_v^t, and (iii) the ECN marking rate ECN_v^t (normalized by the link capacity). Then, agents construct their corresponding input feature vector x_v^t based on the current values of these metrics as well as the values of the previous p steps, i.e.

$$x_v^t = (u_v^t, q_v^t, ECN_v^t, u_v^{t-1}, q_v^{t-1}, ECN_v^{t-1}, \dots, u_v^{t-p}, q_v^{t-p}, ECN_v^{t-p}).$$

Empirically we found that $p = 2$ works best for providing some temporal context while keeping a low input dimension.

Agents initialize their initial Message Passing Neural Network (MPNN)-based hidden state h_v^0 with their input feature vector x_v^t; since the hidden state vector dimension is typically equal or higher than that of the input feature vector, remaining components are simply 0-padded. MAGNNETO-CC then executes K message-passing steps in which distributed communications between agents are involved. More in detail, at each message-passing step k each agent $v \in \mathcal{V}$ first sends its current hidden representation h_v^k to its egress neighbors $\mathcal{B}_e(v)$, and consequently they receive a set of messages from their ingress adjacent agents $\mathcal{B}_i(v)$. At that point, agents individually combine each of the received hidden states with their own through the message function –in our case, a feed-forward neural network–, and all the resulting outputs are in turn merged into a fixed-size representation M_v^k via the aggregation function. For doing so, we implement element-wise operations, such as min and max. Finally, each agent applies the update function –another fully-connected neural network– to the aggregated information M_v^k and its own hidden state h_v^k, which outputs its new hidden representation h_v^{k+1}.

After concluding the K message-passing steps, all agents end up with a final hidden state h_v^K for that specific time-step t. This final representation is then fed by each agent into their readout module, which provides with the final values that are used to define their individual policies. In particular, since MAGNNETO-CC implements a Q-learning based pipeline, each agent's readout directly outputs the q-value estimates of all possible actions for the current state of the DCN, and as described in previous Subsection 5.3.1 each agent takes the action with the highest value. This lead us to the actual definition of the action space. MAGNNETO-CC faces the CC problem by optimizing the ECN marking thresholds, and by design it is able to adapt the ECN parameters $\{k_{min}, k_{max}, p_{max}\}$ for each individual switch egress port. In our implementation, we discretize the values of these parameters into some pre-defined values; we provide more details in Section 5.4.1.

Regarding the learning process, all MPNN internal modules (message, update and readout functions, which are replicated among all agents) are trained based on the rewards computed at each step t of a training episode. More in detail, in our implementation we define the reward r_v^t of agent $v \in \mathcal{V}$ at step t as

$$r_v^t = w_1 \cdot f(q_v^t) + w_2 \cdot u_v^t,$$

where $f(\cdot)$ is a decreasing function with respect to the queue length q_v^t of the associated port ($\mathrm{dom}(f) = [0, 1]$, based on [24]), u_v^t accounts for the utilization of that port, and $w_1, w_2 \in [0, 1]$ are the corresponding weights, with $w_1 + w_2 = 1$ ($w_1 = 0.7$ and $w_2 = 0.3$ worked best in our experiments). At each time step t of a training episode, MAGNNETO-CC gathers the global RL-based sequence $(\{x_v^t\}_{v\in\mathcal{V}}, \{r_v^t\}_{v\in\mathcal{V}}, \{x_v^{t+1}\}_{v\in\mathcal{V}})$ and stores it as a single sample in a replay buffer. Then our model randomly selects a batch of these samples from the buffer and performs the training and update of Q_θ accordingly [48].

5.3.3 Discussion on Deployability

In this section we discuss some relevant practical implications when deploying the proposed solution in DCNs. MAGNNETO-CC naturally distributes the modules of the MPNN among the switches of the DCN, which enables to parallellize all the node-level computations and communications on the local neighborhood. By construction, the GNN-based modules behind MAGNNETO-CC –interpreted as agents– can be replicated and deployed in any switch, to optimize the ECN configuration on a particular NIC, regardless of the size and shape of the DCN topology considered. The process of MAGNNETO to exchange messages between neighbors –i.e. share the agent's hidden states– is in fact generic and scale-invariant [131]. This provides our solution with excellent scalability and generalization capabilities.

MAGNNETO-CC applies parameter sharing over all the NIC-based agents; thus during training the MPNN modules jointly learn from the individual perspective of all NICs in the network, and also in this process agents learn what information to exchange with neighbors in order to achieve effective coordination between agents at different levels of granularity –following the topology structure from the local context (i.e., direct neighbors) to a more global context within the network. Note, however, that no matter all agents are in fact replicas, at execution time each of them is able to specifically adjust its behaviour based on its local state and the information received from its neighbors.

Finally, the multi-agent formulation of MAGNNETO facilitates exploring the large solution space. By enabling the adjustment of ECN parameters $\{k_{min}, k_{max}, p_{max}\}$ at the interface level, the combinatorial of all the possible actions would explode from a single-agent perspective –specially taking into account that modern datacenters may have up to tens of thousands of servers [127,132]. In this sense, MAGNNETO-CC approach of defining a policy for each of these NIC instances separately allows to effectively deal with such complexity and fully distribute the agent's decision making.

5.4 Evaluation

This section is devoted to analyze the experimental results of MAGNNETO-CC. First, we describe the considered baselines as well as the setup used in our evaluations.

5.4.1 Experimental Setup

We use the ns-3 simulator [133]. Experiments are done in a 2-layer Clos Network similar to the testbed used in ACC [24], with 24 hosts, 4 leaf switches, and 2 spine switches. Switch-to-switch links have a capacity of 100Gbps, and host-to-switch links have 25Gbps. All links have a propagation delay of 1μs, consequently the maximum base RTT is 8μs. Switches have a shared-memory buffer of 32MB, which is derived from real devices [124]. The network comprises a single RDMA domain, and PFC is enabled on switches [134]. End-hosts implement a DCQCN [100] distribution mimicking the implementation of Mellanox ConnectX-4 cards [103], with a fixed window that limits the inflight bytes to the maximum Bandwidth-Delay Product (BDP) in the network. This configuration is known to perform better than the original one, as it avoids PFC storms [103]. During our simulations, MAGNNETO-CC agents act every 100μs, and each episode lasts 25ms. We then leave sufficient time to ensure the same set of flows finishes in all experiments, including the longest flows.

We note that we will always consider the same MAGNNETO-CC model along all our evaluation, which was exclusively trained using a real-world *FB_ Hadoop* workload [127] trace with a normal traffic load of 60% [103] and considering periodic 16:1 incasts events [24]. In our implementation of Q_θ, message, update and readout functions are 2-layer feed forward

NNs, and the aggregation function combines an element-wise *min* and *max*. Regarding the tuning of some important hyper-parameters of the model, the dimension of the agents' hidden states h_v is set to 24, and $K = 2$ message passing steps are considered —experimentally, we have found that more iterations do not lead to better performance, since the diameter of DCNs is typically very limited (2-3 hops). In addition, the possible ECN configurations are discretized according to the sets $k_{min} = \{2, 4, 8, 16, 32\}$, $k_{max} = \{16, 32, 64, 128, 256\}$ and $p_{max} = \{0.01, 0.25, 0.5, 0.75, 1.0\}$, resulting in an agent action space $\mathcal{A}_v$ with 120 effective combinations.

5.4.2 Benchmarks and Performance Metrics

Before presenting our experimental results, in this section we aim to define how we actually quantify the performance of MAGNNETO-CC. First of all, we considered as baselines the following two state-of-the-art methodologies —revisited in Section 5.5— compatible with widely-deployed ECN protocols:

- DCQCN [100]: This benchmark represents the most widely used approach in today's DCNs, i.e., careful selection of static ECN parameters. It implements the static ECN configuration used in Alibaba's production DCNs [103]:
  $k_{min} = 100KB \times \frac{Bw}{25Gbps}$; $k_{max} = 400KB \times \frac{Bw}{25Gbps}$.
- ACC [24]: State-of-the-art solution for ECN tuning based on MARL (without GNN). We have implemented it based on the description in [24]. This solution is designed for online training and it does not perform parameter sharing across agent implementations (i.e., agents are trained independently). For fairness, we re-train the solution on each evaluation scenario selecting the best set of agents after hyper-parameter tuning.

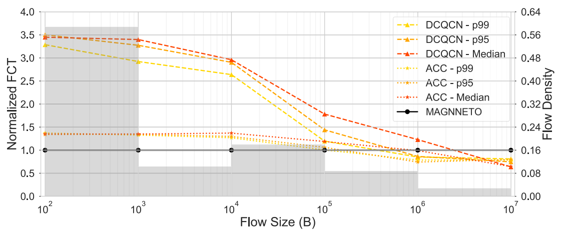
Our goal is to evaluate our trained model in varying DCN scenarios —most of them different than those seen in training— and compare the results against these benchmarks. As stated in Section 5.2, the FCT is broadly accepted the most complete performance metric in DCNs, and we pay special attention to the 95 and 99 percentiles of this metric due to the aforementioned impact on the partition/aggregate design patterns dominant in today's applications [99, 103]. Hence, for each scenario, we compute the FCT median, 95-pct and 99-pct obtained by our solution against those of the benchmarks. In particular, we designed a detailed visualization where *(i)* results are aggregated and shown by flow size; *(ii)* the FCT values of the baselines are normalized with respect to those of our MAGNNETO-CC model; and *(iii)* the flow size distribution is shown in parallel to properly contextualize the relevance of the obtained results. Finally, these plots are complemented with a direct table comparison of the mean absolute values of FCT slowdown, throughput and queue length metrics.

5.4.3 Direct Performance Comparison

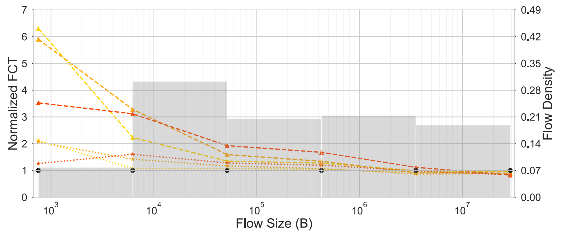
In our first set of experiments, we aim to evaluate our solution in the same scenario considered in training, with traffic generated from the *FB_ Hadoop* workload. We generate two different traffic traces to test the performance of our method: with periodic incasts –similar to those seen during training– and without them. Respectively, Figures 5.4a and 5.4b compare the per-flow size FCT median, percentile 95 (p95) and percentile 99 (p99) achieved by MAGNNETO-CC against the defined baselines.

In both cases, we can see that MAGNNETO clearly outperforms the static DCQCN setting and improves the state-of-the-art ACC solution, especially on short and medium sized flows –which represent the vast majority of flows– in median and both p95 and p99 tails. An aspect that we can observe in this case is that MAGNNETO-CC learns to slightly sacrifice latency for long flows, as they account for a small percentage of the total counting. Overall, and as shown in Table 5.1, MAGNNETO achieves a notable reduction in the mean FCT slowdown (up to 16.6% reduction with respect to ACC without incasts, and 12.6% reduction without) while keeping equivalent mean throughput to that of ACC (1.38% of difference in the worst case).

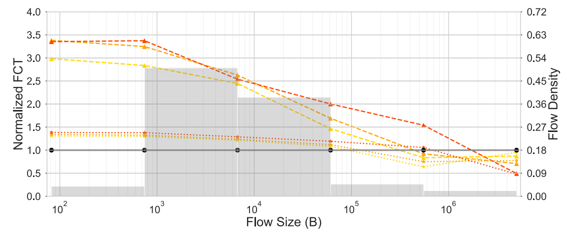
Moreover, Table 5.1 also shows an interesting behaviour of our solution: it achieves a significant reduction in queue length –more than 50% in both *FB_ Hadoop* experiments–, which directly relates to significantly lower buffer occupancies in switches. These results suggest that MAGNNETO-CC is able to attain good flow-level latency (i.e., FCT) by learning how to efficiently manage queues so as to achieve stability and be prepared for potential microbursts and incast events. This near-zero queue behavior is a trend seen in other state-of-the-art solutions, such as HPCC [103] (based on advanced telemetry), and it especially helps achieve ultra-low latency on short flows. As previously discussed in Section 5.2, this behavior is also very beneficial in modern DCNs, given the ongoing trend on decreasing the ratio between buffer size and switch capacity.



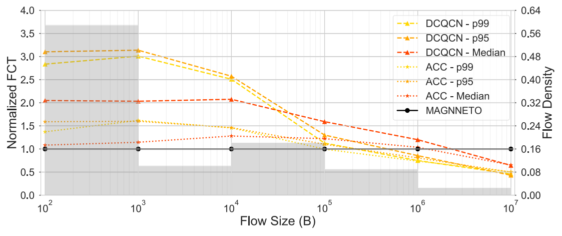
(a) *FB_ Hadoop* with incasts



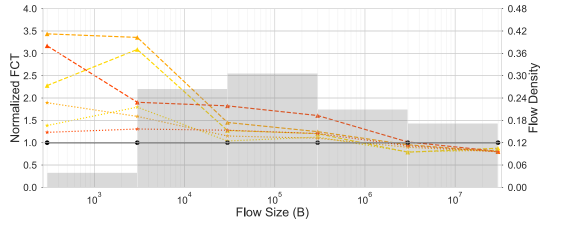
(c) *WebSearch* with incasts



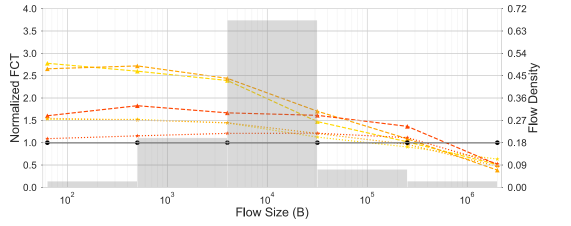
(e) *AliStorage* with incasts



(b) *FB_ Hadoop* without incasts



(d) *WebSearch* without incasts



(f) *AliStorage* without incasts

Figure 5.4: Normalized FCT median, p95 and p99 of the DCQCN and ACC baselines –with respect to MAGNNETO-CC– when considering different traffic workloads (*FB_ Hadoop*, *WebSearch*, *AliStorage*) with 16:1 incasts (top) or without (bottom). Our MAGNNETO model is exclusively trained on traffic from *FB_ Hadoop* with incasts. The measured flow density can be visualized to facilitate the interpretation of the results.

		Mean FCT Slowdown				Mean Throughput (Mbps)				Mean queue length (kB)			
		DCQCN	**ACC**	**MAGNNETO**		**DCQCN**	**ACC**	**MAGNNETO**		**DCQCN**	**ACC**	**MAGNNETO**	
Incasts 16:1	*FB_Hadoop*	5.76	3.25	**2.84**	↓ -12.6%	**401**	399	398	↓ -0.25%	42.4	12.6	**6.00**	↓ -52.4%
	WebSearch	6.48	5.38	**5.30**	↓ -1.48%	**395**	394	390	↓ -1.02%	37.3	9.64	**1.65**	↓ -82,9%
	AliStorage	5.95	3.49	**3.10**	↓ -11.2%	399	**400**	399	↓ -0.25%	49.2	16.7	**9.70**	↓ -41,9%
No Incasts	*FB_Hadoop*	3.94	2.59	**2.16**	↓ -16.6%	**362**	361	356	↓ -1.38%	26.6	10.0	**3.83**	↓ -61.7%
	WebSearch	4.37	3.70	**3.65**	↓ -1.35%	345	**346**	343	↓ -0.86%	24.2	7.95	**1.14**	↓ -85,7%
	AliStorage	3.72	2.71	**2.38**	↓ -12.2%	**355**	**355**	354	↓ -0.28%	27.0	13.2	**6.73**	↓ -49.0%
Big loads	High (70%)	5.81	3.41	**2.80**	↓ -17.8%	**416**	415	411	↓ -0.96%	42.0	12.5	**5.34**	↓ -57.2%
	Extreme (80%)	7.01	4.27	**3.41**	↓ -20.1%	**444**	443	438	↓ -1.13%	50.8	15.4	**6.38**	↓ -58.6%

Table 5.1: Mean FCT slowdown, throughput, and queue length obtained by MAGNNETO-CC and the baselines for different traffic workloads (*FB_Hadoop*, *WebSearch*, *AliStorage*) —with and without incasts—, and with high traffic loads. It also includes the relative difference (in %) of MAGNNETO results with respect to those of ACC.

5.4.4 Evaluation under Traffic Changes

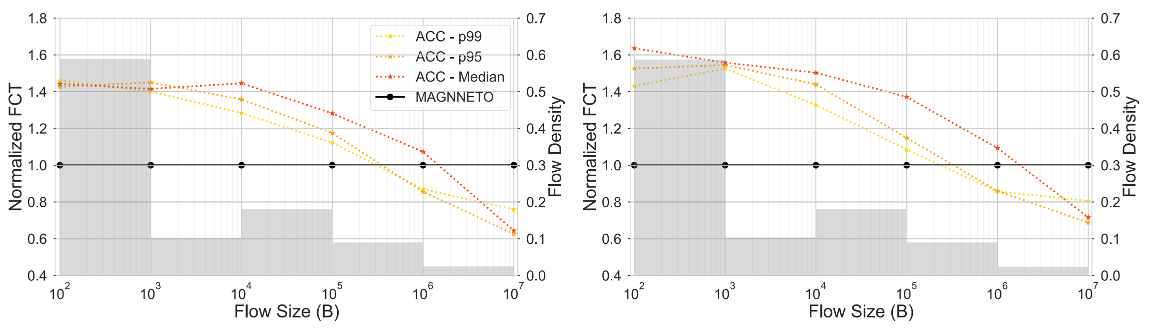
Our next goal is to demonstrate that our trained MAGNNETO-CC model can adapt to unseen traffic scenarios without requiring any further re-training. In this regard, we design two different challenging scenarios: different workload distributions, and very high traffic loads.

Different Traffic Workloads

Different applications may have completely different traffic distributions, so we are interested in testing how our trained MAGNNETO-CC model performs over workloads not previously seen in training. Hence, in this set of experiments we generate traffic traces from two different real-world traffic distributions, *WebSearch* and *AliStorage*, with and without periodic incasts. Figure 5.4 and Table 5.1 summarize the experimental results. We first notice that the flow size distributions of these workloads greatly differ from the one seen in training –i.e. *FB_Hadoop*. In particular, *WebSearch*-based traffic (see Figures 5.4c and 5.4d) involves dealing with a considerable higher amount of medium and long-sized flows. However, we can observe that MAGNNETO-CC improves its FCT metrics for the longer flows on this scenario, suggesting that it is properly prioritizing them. In fact, as shown in Table 5.1, MAGNNETO-CC still gets a slightly better mean FCT slowdown than ACC in this case, with and without incast bursts. On the other hand, MAGNNETO-CC model again sacrifices a bit of long flows' latency for AliStorage traffic traces (Figures 5.4e and 5.4f), since in this scenario those flows have even less density than in *FB_Hadoop* traces. By doing so, our model is able to obtain better FCT median, p95 and p99 metrics than baselines for the major number of flows –with and without incasts–, and reducing the mean FCT slowdown of ACC by more than 11% (Table 5.1). As we can also see in Table 5.1, MAGNNETO-CC achieves such good FCT-based results in all these scenarios only at the expense of a small reduction in the mean throughput (1% of difference with respect to ACC at worst). However, MAGNNETO-CC provides as well with significant improvements in terms of buffer occupancies, lowering ACC's mean queue length from $40 - 50\%$ for AliStorage traffic, up to more than 80% for *WebSearch* traces.

Higher Traffic Loads

Next, we analyze whether our trained model can successfully operate over higher traffic loads. For doing so, we evaluate our MAGNNETO model over new traces from the *FB_-Hadoop* workload with an average network load of 70% and 80% –instead of the average 60% experienced in training. Figure 5.5 presents the normalized FCT median, p95 and p99 achieved by ACC with respect to our solution in both cases. We can see both in Figure 5.5 and in Table 5.2 that the results of both loads are alike, and not very different to those previously observed with a 60% load: compared to ACC, MAGNNETO-CC improves all three FCT metrics for short and medium flows –which represent almost 90% of the total



(a) Very high load - 70% load (b) Extreme load - 80% load

Figure 5.5: Evaluation of MAGNNETO-CC against ACC when considering higher loads than those seen in training (*FB_Hadoop* workload).

	FCT Slowdown		**Throughput (Mbps)**		**Queue Length (kB)**	
	ACC	**MAGNNETO**	**ACC**	**MAGNNETO**	**ACC**	**MAGNNETO**
1 failure	2.91	**2.66** (-8.59%)	**343**	338 (-1.46%)	9.10	**3.89** (-57.3%)
2 failures	5.80	**5.72** (-1.38%)	**275**	275 (=)	7.92	**4.91** (-38.0%)
Extra branch	1.76	**1.56** (-11.3%)	**543**	541 (-0.37%)	5.73	**2.94** (-48.7%)
32 hosts	2.74	**2.20** (-19.7%)	**475**	468 (-1.47%)	10.7	**3.80** (-64.5%)
40 hosts	2.70	**2.27** (-15.9%)	**564**	560 (-0.71%)	8.60	**3.65** (-57.6%)

Table 5.2: Mean FCT slowdown, throughput, and queue length of MAGNNETO-CC and ACC under various topological changes: link failures (top), adding an extra branch (middle), and adding extra hosts (bottom). In all cases, traffic is generated from the *FB_Hadoop* workload.

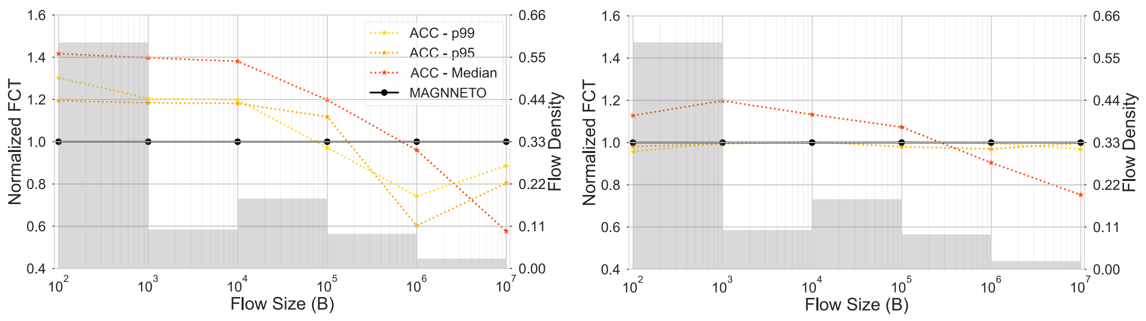
flows–, and achieves equivalent mean throughput –within 1.35%. MAGNNETO manages to reduce the mean queue length by more than 57%. These results suggest that our solution is robust against varying traffic loads.

5.4.5 Evaluation under Topology Changes

Lastly, we define a set of experiments involving topological changes to further evaluate the robustness of a trained MAGNNETO-CC model. In particular, we analyze two different scenarios: random link failures, and flatter networks –i.e. adding extra hosts connected to leaf switches.

Link failures

In production DCNs, link failures occur frequently [108, 109], and often lead to network asymmetries and severe performance degradation (see Section 5.2.2). In these experiments, we simulate how MAGNNETO responds to critical failures in links between leaf and spine switches. Figures 5.6a and 5.6b show, respectively, a comparison of FCT metrics between



(a) 1 link failure (b) 2 link failures

Figure 5.6: Evaluation of MAGNNETO-CC against ACC when experiencing link failures (*FB_ Hadoop* workload).

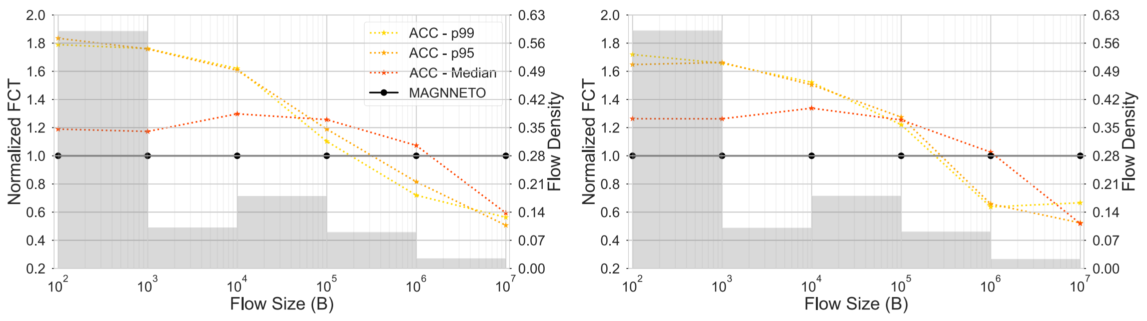
MAGNNETO-CC and ACC, for 1 and 2 link failures. In the case of a single failure, MAGNNETO improves ACC on these metrics for most of the flows. On the other hand, the 2-link failure scenario presents less performance gap between our method and ACC, which was individually trained on each target network scenario given its online nature. Here, our model gets a small improvement in median FCT for small and medium flows, but the p95 and p99 tails are equivalent to those obtained by ACC for all sizes. This exposes how challenging this scenario is, especially for models pre-trained offline, as it is the case of MAGNNETO-CC. Overall, in terms of absolute values (Table 5.2), even with 2 failures the proposed solution still provides with a slightly better mean FCT slowdown serving exactly the same throughput, and does so while reducing the mean queue length up to a 38%.

Network upgrade

Datacenters are periodically upgraded to increase their processing and switch capacity, for example by adding new Points-of-Delivery to the network [24]. This can change drastically the overall network state. In these experiments we aim to simulate a topology upgrade. In particular, we add a new branch to the 2-layer Clos topology considered in our experiments, which includes an additional core switch, two spine switches, and six hosts. In Table 5.2 we observe that MAGNNETO-CC achieves significant improvements in terms of FCT with respect to ACC (11.3%), while keeping the same throughput and considerable queue length reduction (48.7%).

Adding extra hosts

Datacenters are periodically upgraded to increase their processing and switch capacity, for example by adding new Points-of-Delivery to the network [24]. This can change drastically the overall network state. In this section, we aim to evaluate how MAGNNETO-CC operates when it is deployed in network topologies with different properties. For this purpose, we increasingly connect new hosts to leaf switches in the original network where MAGNNETO was trained (with 24 hosts, i.e. 6:1 host/switch ratio). Figure 5.7 gathers a



(a) 32 hosts (8:1 ratio)   (b) 40 hosts (10:1 ratio)

Figure 5.7: Evaluation of MAGNNETO-CC against ACC on flatter networks (*FB_ Hadoop* workload).

comparison of the FCT between MAGNNETO-CC and ACC for two new scenarios (with 32 and 40 hosts). Likewise, Table 5.2 shows the corresponding aggregated mean values. The overall performance comparison w.r.t ACC –FCT, throughput and queue length metrics– is similar to that obtained for the original topology with 24 hosts. We observe an increment on the mean throughput –due to the increase of hosts–, as well as a very slight behaviour difference on the p95 and p99 tails of the FCTs for small flows, if we compare it with the analogous results in the original network (Figure 5.4b). Overall, we see that our solution improves ACC a bit further on these metrics, which suggests that our solution can effectively handle flatter networks with higher congestion levels on the core.

5.5 Related Work

CC has been largely studied in the past, with a rich body of proposals especially focused on high-speed DCNs. This section comprises an overview of relevant works related to MAGNNETO-CC.

Advanced CC mechanisms for DCNs: Some recent pioneering works have proposed sophisticated CC mechanisms showing outstanding performance in DCNs [99–105]. For example, HPCC [103] achieves remarkably short FCTs while offering good throughput and stability to traffic changes (e.g., incast events). To this end, it leverages accurate fine-grained measurements produced by modern In-band Network Telemetry (INT) mechanisms [118, 121]. TIMELY [101] and Swift [104] rely on accurate delay measurements on NICs to control flow rates. Other well-known works, such as pHost [116] or Homa [102] are credit-based solutions, where receivers control the flow rate by sending credit packets to senders. All the previous CC mechanisms rely on novel network architectures and/or protocol stacks that unfortunately are not supported by most legacy switches deployed in DCNs nowadays. At the time of this writing, the most widely deployed CC standards are DCTCP [99] in networks running the TCP/IP stack, and DCQCN [100] in RDMA-based networks. Both are ECN-based mechanisms, where switches mark packets in case they detect congestion (i.e., queue length above certain thresholds). MAGNNETO-CC is designed to inter-operate with traditional ECN-based mechanisms in an efficient and distributed way. Although in this dissertation we test MAGNNETO-CC only in RDMA networks with DCQCN [100], our solution can be easily adapted to optimize any other ECN-based CC mechanism, such as the aforementioned DCTCP [99], or TCP-Bolt [135]).

Machine Learning-based mechanisms for CC: Recent works posit the use of ML techniques to produce data-driven solutions that can efficiently adapt to the network dynamics, which extends the potential to optimize performance in DCNs. Among the most popular solutions, works such as Aurora [112], or Orca [113], propose to use RL to adapt flow rates at end-hosts. These solutions are focused on adapting the flow rate at end-hosts according to the congestion feedback received, and they require to re-implement the network stack. Likewise, AuTO [111] proposes a novel two-level mechanism that accurately controls routing for long flows and queue scheduling for optimizing the latency of short flows. Instead, MAGNNETO-CC is focused on distributed in-network optimization of widely deployed ECN-based CC mechanisms. The closest work to MAGNNETO-CC is arguably ACC [24], where the authors propose a MARL-based mechanism to optimize ECN-based schemes. However, that work — and all previous ones — contemplate online training to dynamically learn how to adapt to the network state. As a result, these ML-based solutions may suffer from transient performance degradations when changes occur in the network (e.g., new traffic workload, incast events, failures). Also, online training entails an intrinsic uncertainty on what would be the resulting performance after re-training. This would require strong supervision mechanisms to check the evolution of agents and be ready to deploy backup mechanisms.

On the other hand, MAGNNETO naturally exhibits high robustness to generalize across traffic and topology changes unseen during the training phase. This is thanks to the GNN architecture that it internally implements, which naturally induces a cooperation mechanism among agents —via message exchanges between neighbours– that contrasts with the greedy behaviour of ACC agents —which act based on its local measurements, as no contextual data is provided to them. On top of that, our GNN design implements *parameter sharing* to finally produce a general agent implementation jointly learned from the individual perspective of each node in the network. This property can be especially interesting from a practical standpoint, as it enables to train the solution offline (e.g., in a controlled testbed), and then be able to deploy it directly in production networks, without the need for (re)training it on premises. Also, it permits extensive testing prior to deployment, giving vendors the possibility to issue certifications with the safe operational ranges that the solution would safely support once deployed (e.g., link capacities, max. network size). This process would be better aligned with the standard way network products are commercialized nowadays.

5.6 Discussion

This chapter has introduced MAGNNETO-CC, a distributed solution for in-network CC optimization in DCNs that is compatible with any network running widely deployed ECN-based CC mechanisms, such as DCQCN, or DCTCP. In this MAGNNETO-based architecture, agents are deployed in switches; they cooperate and exchange information to dynamically adapt the ECN configuration and optimize the global flow-level performance. In our evaluation, we have benchmarked MAGNNETO-CC against two baselines: (*i*) a static ECN configuration used in Alibaba's production DCNs, and (*ii*) ACC, a state-of-the-art ML-based solution for dynamic ECN tuning. The experimental results show that our solution significantly outperforms the two previous baselines in terms of FCT. At the same time, we have observed that our solution learns to keep small queue lengths (up to 85.7% with respect to ACC), thus being especially interesting for next-generation DCNs, where buffer size is expected to continue shrinking with respect to the skyrocketing switch capacities.

As discussed earlier, an important feature of ML-based solutions for networking is their capability to generalize to different scenarios to those seen during training, as it avoids the need for online training. However, existing ML-based CC optimization solutions are designed to be trained online and gradually learn how to adapt to the current network conditions. In this vein, MAGNNETO-CC exhibits good behavior when operating in new scenarios never seen during training, such as shifts on the traffic workload, or topology changes. This is thanks to its internal GNN-based framework, which leverages two main features: (*i*) during training MAGNNETO produces a single agent implementation jointly learned from the individual perspective of each agent in the network, using *parameter sharing*, and (*ii*) agents run a topology-aware message passing mechanism to get local context and cooperate with each other. As a result, MAGNNETO-CC produces more robust and general agent implementations that can successfully operate on significantly different network scenarios to those seen during training.

Future Directions & Conclusions

Chapter 6

Beyond the Graph Domain: Topological Network Traffic Compression

As shown in previous chapters, the MAGNNETO architecture has demonstrated remarkable capabilities for distributed optimization in networked scenarios. By leveraging Multi-Agent Reinforcement Learning (MARL) and Graph Neural Network (GNN), our proposed method has been capable to exploit and process relational data that naturally arises in the graph domain –computer networks, power grids–, while at the same time is able to parallellize the optimization process through the network due to its modular-based design. The combination of these two features has allowed MAGNNETO to achieve a great trade-off between performance, execution cost, and also generalization over scenarios not previously seen during the training phase.

However, in the last part of my thesis we wanted to consider possible future directions to extend and/or transform MAGNNETO to handle even more general and challenging scenarios. In particular, given the current ongoing and active research about the over-smoothing and over-squashing issues of GNN models [136, 137], our idea was to explore how to improve our solution to properly undertake long-range interactions that might rise through the considered networked scenario.[1]

In this chapter we present some preliminary research about this topic. In our aim to extend our previous work to naturally encompass arbitrary interactions beyond the local neighborhoods defined by graph representations, we found out that the relatively recent ML branch of Topological Deep Learning (TDL) [30] could be of great help in this regard.

[1]Over-smoothing and over-squashing problems become more crucial as the depth of GNN architectures increases (i.e. with more message passing iterations), which particularly seems to prevent current GNNs models from detecting and fully exploiting long-range interactions [137].

In fact, TDL methods take GNNs architectures a step further by working on domains that can feature higher-order relations. By leveraging (algebraic) topology concepts to encode multi-element relationships (e.g. simplicial [138], cell [139] and combinatorial complexes [31]), Topological Neural Network (TNN) architectures allows for a more expressive representation of the complex relational structure at the core of the data. Despite its recent emergence, TDL is already postulated to become a relevant tool in many research areas and applications [31], including complex physical systems [140], signal processing [141], network science [142], molecular analysis [138] and social interactions [143]. molecular classification and design [138, 144] and social interactions [143], to name a few.

In particular, the methodology that we introduce in this chapter can be interpreted as a first re-design of MAGNNETO's architecture where the GNNs that model the environment are replaced by TNNs instead. Despite the fact that the current proposal do not combine –yet– TNNs methods with MARL, TDL-inspired architectures can also be understood as a multi-agent optimization techniques due to the intrinsic modularity of TNNs: as GNNs, TNNs also follow a message-passing scheme that can be naturally distributed, even if the interactions no longer rely on the original network topology.

To motivate the potential improvements of these novel methodologies, we considered the challenging use case of lossy data compression, with a particular emphasis in data coming from networked scenarios as well. Although this data might be naturally represented in the graph domain, we argue that this task can hugely benefit from TDL by enabling to exploit multi-way correlations between possibly distant elements in the network (e.g. generator and sink nodes in computer networks). Motivated by this, we propose in this chapter a novel TDL framework to *(i)* first detect higher-order correlated structures over a given data, and *(ii)* then apply TNNs to obtain compressed representations within those multi-element sets. The goal is to show that TDL-inspired approaches might be more suitable to perform data compression than other ML-based architectures –and in particular GNNs.

The organization of the chapter is as follows. First, in Section 6.1 we motivate and describe the considered use case: network traffic compression in networks with multiple vantage points. Section 6.2 details the proposed topological-inspired methodology to perform (graph) signal compression. Then Section 6.3 shows the evaluation, in which we compared our methods to other ML-based methodologies, and Section 6.4 reviews the related work. The chapter concludes with a final discussion in Section 6.5.

6.1 Network Scenario

Computer Network's traffic has significantly increased in recent years [145], specially driven by the development of new applications –such as vehicular networks, Internet of Things, virtual reality, video streaming– and the advancement of network technology –e.g. the fast improvements in link speed. In fact, current Internet Service Provider (ISP) and

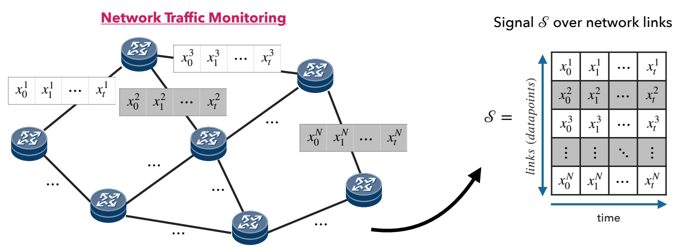

Network Traffic Monitoring
Signal $\mathcal{S}$ over network links
$\mathcal{S}$ =
links (datapoints)
time

Figure 6.1: Goal: to compress a signal $\mathcal{S}$ over a network representing the temporal evolution of each link utilization.

datacenter networks can easily produce hundreds of terabytes of traffic traces per day [127], and this keeps growing.

However, network operators continuously need to store and analyze network traffic data for various network management purposes, including network planning, traffic engineering, traffic classification, anomaly detection or network forensics. With those huge amounts of generated data, the efficient storage of all this information is then becoming a crucial aspect for them [146].

This is what motivated us to choose network traffic compression for testing our proposed method. Not only it provides with complex data naturally represented in the graph domain, but also represents a relevant use case for the networking community. In addition to that, given that most –if not all– network management tasks admit a reasonable loss tolerance, it makes sense to consider lossy methods such as the aforementioned *zfp* [147] or ours. Another relevant fact that motivated the choice of this use case is that there exists public real-world datasets [148] that, despite of their limited network sizes, already reflect complex traffic patterns that may go beyond the provided graph structure (e.g. with distant elements possibly having strong correlations, such as links that are adjacent to generator and sink nodes).

In particular, in our evaluation we consider two real-world backbone networks's datasets from [148] –Abilene and Geant, more details in Section 6.3.1–, and target the problem of compressing the temporal per-link traffic evolution (Figure 6.1). Once the original link-based signal is divided into processable temporal windows, we benchmark our method against a curated set of GNN-based architectures –and a Multi-Layer Perceptron (MLP)– properly designed for compression as well. Obtained results clearly suggest that our topological framework defines the best baseline for *lossy neural compression*.

6.2 Proposed Method

This section describes the proposed Topological (Graph) Signal Compression framework, which is divided into the following three primary modules:

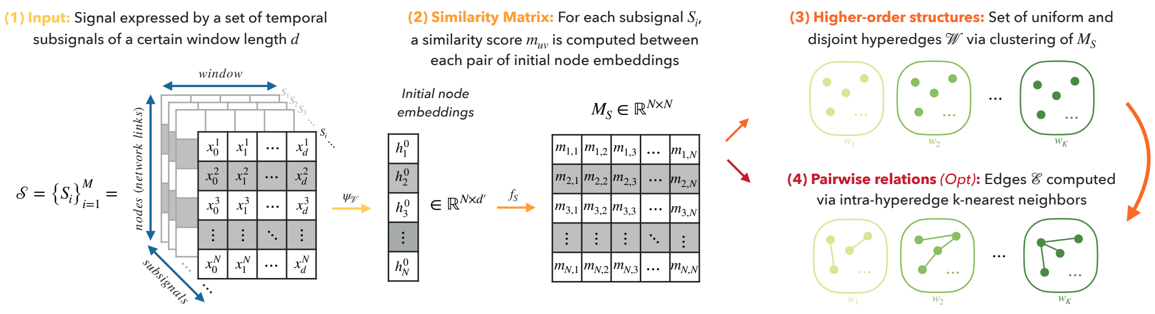

(1) Input: Signal expressed by a set of temporal subsignals of a certain window length d
window
nodes (network links)
subsignals
$\mathcal{S} = \{S_i\}_{i=1}^M =$
$\psi_\mathcal{V}$
(2) Similarity Matrix: For each subsignal S_i, a similarity score m_{uv} is computed between each pair of initial node embeddings
Initial node embeddings
$\in \mathbb{R}^{N \times d'}$
f_S
$M_S \in \mathbb{R}^{N \times N}$
(3) Higher-order structures: Set of uniform and disjoint hyperedges $\mathcal{W}$ via clustering of M_S
(4) Pairwise relations (Opt): Edges $\mathcal{E}$ computed via intra-hyperedge k-nearest neighbors
w_1
w_2
w_K

Figure 6.2: Topology Inference Module. For each subsignal $S_i \in \mathcal{S}$, it outputs a topological object $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ determined by K disjoint hyperedges.

6.2.1 Topology Inference Module

The first stage of the proposed model infers the computational topological structure –both pairwise and higher-order relationships– from the data measurements. In general, the framework assumes to have a set $\mathcal{S}$ of M signals, $\mathcal{S} = \{S_i\}_{i=1}^M$, where S_i consists of N vector-valued measurements x_j of a pre-defined dimension d, i.e. $S_i = \{x_j^i\}_{j=1}^N$, $x_j^i \in \mathbb{R}^d$. Thus, the pipeline that we describe as follows (also shown in Figure 6.2) is independently applied to every signal S_i.

Similarity Matrix: The initial signal $\{x_j\}_{j=1}^N$ [2] is encoded with a MLP into an embedding space

$$h_j^0 = \psi_{\theta_\mathcal{V}}(x_j) \in \mathbb{R}^{d'}, \ \forall j \in \{1, \ldots, N\}.$$

Next, we compute the pairwise similarity matrix $M_S = (m_{uv}) \in \mathbb{R}^{d' \times d'}$ where $m_{uv} := f_S\left(h_u^0, h_v^0\right)$ and $f_S : \mathbb{R}^{d'} \times \mathbb{R}^{d'} \rightarrow \mathbb{R}$ is a similarity function.

Higher-order Relationships: We use clustering techniques on the similarity matrix M_S to deduce K higher-order structures, over which a topological Message Passing pipeline –see next Section 6.2.2– performs the compression. In fact, the idea is to compress the signal within the inferred multi-element sets and encode compressed representations of the data into the final hidden states of these hyperedges. Therefore, the number of higher-order structures K is desired to be considerably lower than the number N of datapoints ($K \ll N$); we design the following *clustering* scheme for this purpose:

1. The number of hyperedges are defined as $K = \lfloor N/p \rceil$, where p is a hyperparameter that identifies the maximum hyperedge length.

2. For every row in the similarity matrix M_S, we extract the top $p-1$ highest entries and calculate their sum. We then select the row that corresponds to the highest summation

[2] For the sake of simplicity, and as abuse of notation, we will avoid writing the superscript i when referring to the measurements of a generic signal $S_i \in \mathcal{S}$.

value. This chosen row becomes the basis for forming a hyperedge as we gather the indices of the $p - 1$ selected columns along with the index of the row itself. Then the gathered indices are removed from the rows and columns of the similarity matrix M_S, obtaining a reduced $\hat{M}_S \in \mathbb{R}^{(d'-p)\times(d'-p)}$.

3. Previous step 2 is repeated with subsequent $\hat{M}_S$ until K disjoint hyperedges are obtained.[3]

On the other hand, the choice of the similarity function becomes a crucial aspect for the compression task. Supported by our early experiments (see Section 6.3), our framework makes use of the **Signal to Noise Ratio (SNR)** distance metric presented in [149], proposed in the context of deep metric learning as it jointly preserves the semantic similarity and the correlations in learned features [149].

Pairwise relationships: Besides higher-order structures, our framework can optionally leverage graph-based relational interactions, either *(i)* by considering the original graph connectivity if it is known, or *(ii)* by inferring the edges via the similarity matrix as well –by connecting each element with a subset of top k row-based entries in M_S. In our experiments only intra-hyperedge edges have been considered to keep the inferred higher-order structures completely disjoint from each other.

6.2.2 Compression Module via Topological Message Passing

We implemented two topological Message Passing (MP) compression pipelines, named SetMP and CombMP. SetMP is a purely set-based architecture that operates only over hyperedges and nodes; CombMP, our most general architecture, leverages the three different structures (nodes, edges, hyperedges) in a hierarchical way,[4] and can be seen as a generalisation of SetMP. We describe both solutions as follows:

Remark: We follow the standard notation of a Message Passing Neural Network (MPNN) to describe our Topological message passing scheme: message $\psi_{\theta.}$ and update $\phi_{\theta.}$ operators represent parametrized learnable functions, and $\oplus$ denotes a permutation-invariant aggregator function.

CombMP Architecture

For a given signal S_i and its corresponding initial embeddings $\{h_j^0\}_{j=1}^N$, CombMP operates over a topological object $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ where $\mathcal{V}$ denotes the set of elements or nodes,

[3]When N/p is not an even division, at some point of the process the ranking starts considering the row-wise $p-2$ higher entries to form $p-1$-length hyperedges, so that at the end a total of $K = \lfloor N/p \rceil$ hyperedges of lengths p and $p-1$ are obtained; see 6.3.1 for further details.

[4]Edges and hyperedges are distinguished because, analogously to recent Combinatorial Complexes (CCC) models [31], edges can hierarchically communicate with hyperedges if they are contained in them; in fact, the name CombMP relates to these general topological constructions.

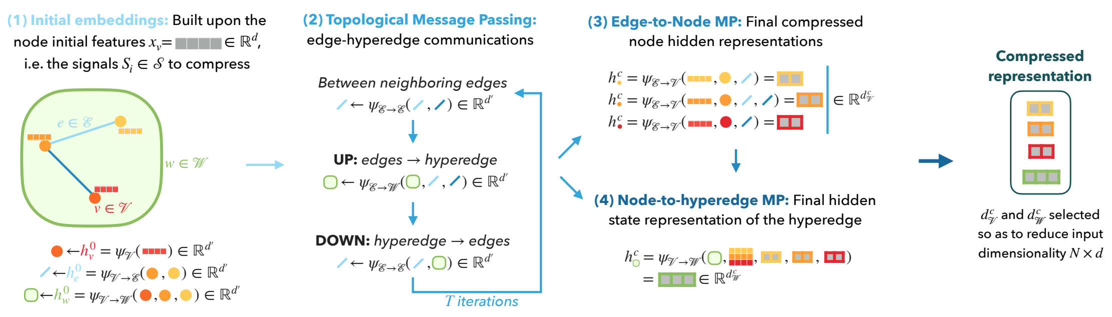

(1) Initial embeddings: Built upon the node initial features $x_v \in \mathbb{R}^d$, i.e. the signals $S_i \in \mathcal{S}$ to compress
$e \in \mathcal{E}$
$w \in \mathcal{W}$
$v \in \mathcal{V}$
(2) Topological Message Passing: edge-hyperedge communications
Between neighboring edges
UP: edges → hyperedge
DOWN: hyperedge → edges
T iterations
(3) Edge-to-Node MP: Final compressed node hidden representations
(4) Node-to-hyperedge MP: Final hidden state representation of the hyperedge
Compressed representation
$d^c_{\mathcal{V}}$ and $d^c_{\mathcal{W}}$ selected so as to reduce input dimensionality $N \times d$

Figure 6.3: Compression Module workflow for the CombMP architecture. It is independently applied to each hyperedge $w \in \mathcal{W}$ of the inferred topological object $\mathcal{T} = (\mathcal{V}, \mathcal{E}, \mathcal{W})$.

$|\mathcal{V}| = N$; $\mathcal{E} \in \mathcal{V} \times \mathcal{V}$ represent the set of edges; and $\mathcal{W} \in (\mathcal{V} \times \cdots \times \mathcal{V})$ the set of hyperedges. The compression pipeline (visualized in Figure 6.3) can be described as follows:

Initial embeddings: First, we generate initial embeddings for the three considered topological structures. For nodes, we use the previously computed embeddings $\{h_v^0\}_{v=1}^N$. For edges and hyperedges, (learnable) permutation invariant functions are applied over the initial embeddings of the nodes they contain; respectively,

$$h_e^0 = \phi_{\theta_{\mathcal{E}}} \left(\oplus_{v \in e} h_v^0 \right)$$

for each $e \in \mathcal{E}$, and

$$h_w^0 = \phi_{\theta_{\mathcal{W}}} \left(\oplus_{v \in w} h_v^0 \right)$$

for each $w \in \mathcal{W}$. The same dimension d' is used for all initial and intermediate hidden representations.

Edge-Hyperedge Message Passing: We define a hierarchical propagation of messages between edges and hyperedges. First, neighboring edges communicate to each other to update their representations; denoting the edge neighbors of an edge $e \in \mathcal{E}$ by

$$\mathcal{N}_e^{\mathcal{E}} := \{e' = (u, v) \in \mathcal{E} | e' \neq e, u \in e \vee v \in e\},$$

its new hidden state becomes

$$h_e^1 = \phi_{\theta_{\mathcal{E}\to\mathcal{E}}} \left(\oplus_{e' \in \mathcal{N}_e^{\mathcal{E}}} \psi_{\theta_{\mathcal{E}\to\mathcal{E}}} \left(h_e^0, h_{e'}^0 \right) \right).$$

Next, hyperedges also update their hidden states based on the updated edge representations according to

$$h_w^1 = \phi_{\theta_{\mathcal{E}\to\mathcal{W}}} \left(\oplus_{e \in \mathcal{E}, e \subset w} \psi_{\theta_{\mathcal{E}\to\mathcal{W}}} \left(h_w^0, h_e^1 \right) \right),$$

for each $w \in \mathcal{W}$. Then the idea is to propagate downwards towards the edges, i.e. from hyperedges to edges,

$$h_e^2 = \phi_{\theta_{\mathcal{W}\to\mathcal{E}}}\left(\oplus_{w\in\mathcal{W}, e\subset w}\psi_{\theta_{\mathcal{W}\to\mathcal{E}}}\left(h_e^1, h_w^1\right)\right);$$

and only between edges again. We note that this whole communication process can be iterated T times.

Edge-to-Node Compression: At this point, we perform a first compression step over the nodes by leveraging the updated edge hidden representations, the initial node embeddings, as well as the original node data as a residual connection. Formally, for each node $v \in \mathcal{V}$ we get a compressed hidden representation

$$h_v^c = \phi_{\theta_{\mathcal{E}\to\mathcal{V}}}\left(\oplus_{e\in\mathcal{E}, v\in e}\psi_{\theta_{\mathcal{E}\to\mathcal{V}}}\left(x_v, h_v^0, h_e^t\right)\right) \in \mathbb{R}^{d_{\mathcal{V}}^c},$$

forcing it to have a much lower dimension than the original signal, i.e. $d_{\mathcal{V}}^c \ll d$.

Node-to-Hyperedge Compression: Finally, a second and last compression step is performed over the hypergraph representations, in this case leveraging a residual connection to the original measurements, the previously computed compressed representations of nodes, as well as the updated hidden representations of hyperedges. More in detail, each hyperedge $w \in \mathcal{W}$ obtains its final compressed hidden representation as

$$h_w^c = \phi_{\theta_{\mathcal{V}\to\mathcal{W}}}\left(\oplus_{v\in\mathcal{V}, v\in w}\psi_{\theta_{\mathcal{V}\to\mathcal{W}}}\left(x_v, h_v^c, h_w^t\right)\right) \in \mathbb{R}^{d_{\mathcal{W}}^c}.$$

In this case, since the number K of hyperedges is set to be much lower than the number of nodes, the dimension $d_{\mathcal{W}}^c$ of these final hyperedge representations is not as critical as that of the nodes.

SetMP

In this subsection we describe the topological MP pipeline of SetMP (Figure 6.4). In contrast to CombMP, this architecture disregards binary connections and consequently operates over a topological object $\mathcal{T} = (\mathcal{V}, \mathcal{W})$, where again $\mathcal{V}$ denotes the set of N nodes and $\mathcal{W} \in (\mathcal{V} \times \cdots \times \mathcal{V})$ the set of K inferred hyperedges. We describe the differences in the compression pipeline of this scenario:

Initial embeddings: Initial embeddings for nodes and hyperedges are generated in the same way: $\{h_v^0\}_{v=1}^N$ for the nodes, and

$$h_w^0 = \phi_{\theta_{\mathcal{W}}}\left(\oplus_{v\in w} h_v^0\right)$$

for each $w \in \mathcal{W}$ for the hyperedges (both of them with dimension d').

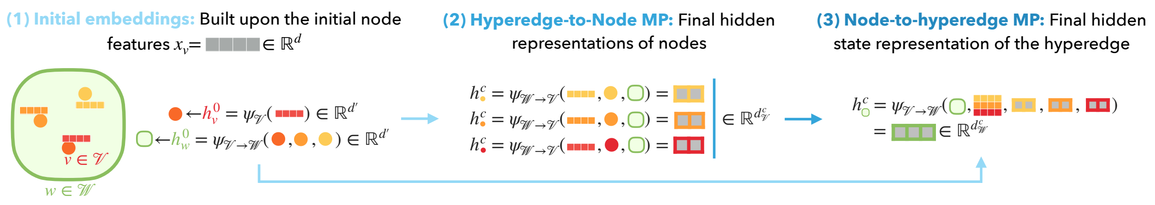

(1) Initial embeddings: Built upon the initial node features x_v= ▨▨▨▨▨ $\in \mathbb{R}^d$
(2) Hyperedge-to-Node MP: Final hidden representations of nodes
(3) Node-to-hyperedge MP: Final hidden state representation of the hyperedge
$v \in \mathcal{V}$
$w \in \mathcal{W}$
● $\leftarrow h_v^0 = \psi_{\mathcal{V}}($▨▨▨▨$) \in \mathbb{R}^{d'}$
▢ $\leftarrow h_w^0 = \psi_{\mathcal{V}\to\mathcal{W}}($●,●,●$) \in \mathbb{R}^{d'}$
$h^c = \psi_{\mathcal{W}\to\mathcal{V}}($▨▨▨▨,●,▢$) =$ ▨▨
$h^c = \psi_{\mathcal{W}\to\mathcal{V}}($▨▨▨▨,●,▢$) =$ ▨▨
$h^c = \psi_{\mathcal{W}\to\mathcal{V}}($▨▨▨▨,●,▢$) =$ ▨▨
$\in \mathbb{R}^{d^c_{\mathcal{V}}}$
$h^c = \psi_{\mathcal{V}\to\mathcal{W}}($▢, ▨, ▨▨, ▨▨, ▨▨$)$
$=$ ▨▨▨ $\in \mathbb{R}^{d^c_{\mathcal{W}}}$

Figure 6.4: Compression Module workflow for the SetMP architecture. It is independently applied to each inferred hyperedge $w \in \mathcal{W}$.

Hyperedge-to-Node Compression: Without edges as intermediaries, we directly perform the node compression, in this case based on both the initial node and hyperedge embeddings and the original signal S_i. Formally, for each node $v \in \mathcal{V}$ we get a compressed hidden representation

$$h_v^c = \phi_{\theta_{\mathcal{W}\to\mathcal{V}}} \left(\oplus_{w \in \mathcal{W}, v \in e} \psi_{\theta_{\mathcal{W}\to\mathcal{V}}} \left(x_v, h_v^0, h_w^0 \right) \right) \in \mathbb{R}^{d^c_{\mathcal{V}}}.$$

Node-to-Hyperedge Compression: The second and last compression step over the hyperedge representations is exactly the same as in the CombMP –except for the fact that now the considered hyperedge hidden states have not been updated. Therefore, each hyperedge $w \in \mathcal{W}$ obtains its final compressed hidden representation as

$$h_w^c = \phi_{\theta_{\mathcal{V}\to\mathcal{W}}} \left(\oplus_{v \in \mathcal{V}, v \in w} \psi_{\theta_{\mathcal{V}\to\mathcal{W}}} \left(x_v, h_v^c, h_w^0 \right) \right) \in \mathbb{R}^{d^c_{\mathcal{W}}}.$$

Compressed Representations

In both architectures, CombMP and SetMP, the final node and hyperedge states, $\left\{ \{h_v^c\}_{v \in \mathcal{V}}, \{h_w^c\}_{w \in \mathcal{W}} \right\}$, encode the compressed representation of a signal $S_i = \{x_j\}_{j=1}^N$. Consequently, the number K of higher-order structures together with the dimensions of those compressed hidden states define the compression factor r_c:

$$r_c = \frac{N \cdot d^c_{\mathcal{V}} + K \cdot d^c_{\mathcal{W}}}{N \cdot d} \tag{6.2.1}$$

6.2.3 Decompression Module

This last module learns to reconstruct the original signal of every node through its compressed representation and the final hidden state of the hyperedge it belongs to. More formally, for each $v \in \mathcal{V}$ and its corresponding hyperedge $v \in w \in \mathcal{W}$, the reconstructed signal $\hat{x}_v$ is obtained as $\hat{x}_v = \phi_{dec} \left(h_v^c, h_w^c \right)$, where ϕ_{dec} is implemented as a MLP in our framework. The whole model is trained end-to-end to minimize the (mean squared) reconstruction error.

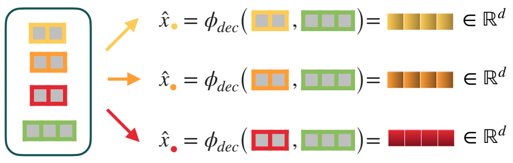

$\hat{x}_{\bullet} = \phi_{dec}(\ ,\ ) = \ \in \mathbb{R}^d$
$\hat{x}_{\bullet} = \phi_{dec}(\ ,\ ) = \ \in \mathbb{R}^d$
$\hat{x}_{\bullet} = \phi_{dec}(\ ,\ ) = \ \in \mathbb{R}^d$

Figure 6.5: Decompression Module. It is applied over each hyperedge-dependent compressed representation set generated by the Compression Module.

6.3 Evaluation

In this section we provide details about how our method has been evaluated.

6.3.1 Experimental Setup

For the evaluation, we use two public real-world datasets –based on Abilene and Geant backbone networks– from [148]. They are pre-processed to generate subsignals S_i of network link-level traffic measurements in temporal windows of length $d = 10$, to which then a random 60/20/20 split is performed for training, validation and test, respectively. In this context, our method is compared against:

- **GNN**: we implemented several standard GNN architectures (GCN [150], GAT [42], GATv2 [44], GraphSAGE [43]) to perform signal compression over the network graph topology; we take the best result among them in each evaluation scenario.
- **MPNN**: a custom MP-based GNN scheme –over the original network graph structure as well– whose modules and pipeline are similar to our proposed topological MPs.
- **MLP**: a feed-forward auto-encoder architecture with no inductive biases over subsignals S_i.

GNN and **MPNN** baselines implement a decompression module similar to that of our TDL-based methods. More technical details are provided below:

Datasets

As previously stated, the traffic traces of both datasets are publicly available at [148]. After distributing them into links, Abilene dataset contains link-level traffic utilization measurements over 6 months –in intervals of 5 minutes– for a topology with 12 nodes and 30 directional links. Considering a temporal window of lenght $d = 10$, this results in $N = 4,809$ subsignal samples after data cleaning. On the other hand, Geant dataset contains analogous measurements for a period of 4 months and a time interval of 15 minutes, in this case for a topology with 22 nodes and 72 directional links. After data cleaning, it has a total of $1,075$ final samples with the same window size.

The topology structure of both networks is also provided, and we use it in our GNN-based baselines. The aforementioned 60/20/20 split is performed over these resulting link-based subsignals.

Implementation of our Proposed Models

Regarding the Topology Inference module, we recall the relevance of the hyper-parameter p that defines the maximum allowed hyperedge length, and which also defines the number of those hyperedges by $K = \lfloor N/p \rceil$, being N the total number of datapoints. In particular, in our implementation we try to form K p-uniform disjoint hyperedges if possible, but otherwise build a combination of $p-1$ and p-uniform disjoint hyperedges –so that every datapoint is contained in one of them. Moreover, we always consider $p > 4$ so that $K \ll N$ holds.

As shown in Equation 6.2.1, this parameter p together with the dimensions of the final node and hyperedge compresed representations, $d^c_{\mathcal{V}}$ and $d^c_{\mathcal{W}}$, define the compression factor of our method. After some hyperparameter tuning, in our experiments we used $p = 8$, $d^c_{\mathcal{V}} = 2$ and $d^c_{\mathcal{W}} = 10$ for achieving $r_c = 1/3$, and $p = 6$, $d^c_{\mathcal{V}} = 5$ and $d^c_{\mathcal{W}} = 10$ for $r_c = 2/3$; respectively, this resulted in 4 and 5 inferred hyperedges for the Abilene dataset, and 9 and 12 for Geant.

These parameters apply to both SetMP and CombMP architectures, and in both scenarios we also consider node, edge and hyperedge hidden representations of dimension $d' = 20$, double the the dimension d of node signals. All message and update functions, $\psi_{\theta_\cdot}$ and $\phi_{\theta_\cdot}$, are implemented as MLPs, and permutation-invariant aggregator functions $\oplus$ consist of the concatenation three element-wise operations: mean, max and min. In the case of CombMP, only 1 iteration of topological message passing (Figure 6.3.2) is performed. Finally, we note that the Decompression Module has the same MLP structure in both compression pipelines.

Baseline Implementations

Analogously to what we do with our proposed architectures, we have fine-tuned the involved hyperparameters of all implemented baselines to perform the compression task. Moreover, they follow a similar compression scheme than our proposed TDL methods. We provide more details below:

Graph-based These methods leverage the original graph-like network structure present in Abilene and Geant datasets. In this case, since the signal is over the edges, we compute the (dual) line graph of the network, so that edges become nodes and are connected between them if they share origin/destination. The idea is then to perform several iterations of message passing over this line graph to get a compressed representation of the original signal in the hidden state of these link-based nodes. The difference between our two graph-based baselines precisely relies on the nature of this message passing:

- **GNN:** Under this name we gather the results of implementing several standard GNN architectures (GCN [150], GAT [42], GATv2 [44], GraphSAGE [43]). In all of these cases, two consecutive message interchanges (with relatively high dimensional hidden states, 64 and 32 in our experiments) are performed before a third one gets the desired compressed representation. We have used the available implementations of PyTorch Geometric [151] for the convolutional GNN layers, and performed an exhaustive hyperparameter-tuning for each of them (testing different hidden dimensions, aggregations, normalizations, dropout values, number of heads, etc.). As previously stated, in each dataset/compression ratio scenario we select the best performing model among this set of GNN architectures to perform the evaluation (shown in Table 6.1). However, we note that there is not a significant difference in performance among the different GNN models, and within a single model different hyperparameter settings do not result in huge performance variations either; we will further expand this analysis in future work.

- **MPNN:** In this case we implement a custom Message Passing GNN whose pipeline resembles that of the CombMP architecture: edges to edges, edges to nodes, nodes to edges, and a final edge to node communication with a residual connection to the original node signal that performs the compression. In this case the intermediate node and edge hidden states' dimension is set to $d' = 20$, just as in our topological-inspired methods. Notably, the implementation of this baseline follows directly from the topological architectures, restricting everything to the graph domain. As a result, it underwent hyperparameter tuning in exactly the same way as CombMP and SetMP.

In both cases, the final hidden states obtained from the last MP step represent the node signal compressed representations. Since the original window-based signals have length $d = 10$, we set this final dimension to 4 and 7 when benchmarking our method against them for getting compression factors r_c of 1/3 and 2/3, respectively. Finally, we note that both graph-based baselines implement a MLP for the decompression task totally analogous to the one of our TDL-based architectures (in this case simply having as input the final node compressed representation).

MLP We also implemented a MLP auto-encoder architecture that considers all possible connections between all elements of each subsignal $S_i = \{x_j\}_{j=1}^{N}$. In particular, each of the subsignals is flattened –i.e. $d_{input} = N \cdot d$– and passed to a feed forward encoder network (with 1024 and 512 hidden dimensions and ReLu activation function after our architecture search) that outputs a final compressed representation of the full subsignal with dimension $d_{output} = \lceil r_c \cdot d_{input} \rceil$, being r_c the considered compression factor. A symmetric decoder network reconstructs the signal from that representation.

Table 6.1: Reconstruction Mean Squared Error (MSE) and Mean Absolute Error (MAE) over the test set of the considered datasets for two different compression factors. **Top:** Considered ML-based baselines. **Middle:** Our proposed topological architectures. **Bottom:** State-of-the-art *zfp* method.

	Abilene				Geant			
	$r_c = 1/3$		$r_c = 2/3$		$r_c = 1/3$		$r_c = 2/3$	
	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
GNN	$1.95 \cdot 10^{-2}$	$1.08 \cdot 10^{-1}$	$1.95 \cdot 10^{-2}$	$1.08 \cdot 10^{-1}$	$2.33 \cdot 10^{-2}$	$1.21 \cdot 10^{-1}$	$2.32 \cdot 10^{-2}$	$1.20 \cdot 10^{-1}$
MPNN	$7.88 \cdot 10^{-4}$	$1.24 \cdot 10^{-2}$	$7.92 \cdot 10^{-4}$	$1.24 \cdot 10^{-2}$	$8.45 \cdot 10^{-3}$	$4.13 \cdot 10^{-2}$	$1.82 \cdot 10^{-3}$	$2.39 \cdot 10^{-2}$
MLP	$1.04 \cdot 10^{-3}$	$1.88 \cdot 10^{-2}$	$9.71 \cdot 10^{-4}$	$1.80 \cdot 10^{-2}$	$3.76 \cdot 10^{-3}$	$3.96 \cdot 10^{-2}$	$3.62 \cdot 10^{-3}$	$3.89 \cdot 10^{-2}$
SetMP	$\mathbf{3.22 \cdot 10^{-4}}$	$\mathbf{8.75 \cdot 10^{-3}}$	$\mathbf{2.03 \cdot 10^{-4}}$	$\mathbf{6.80 \cdot 10^{-3}}$	$\mathbf{6.93 \cdot 10^{-4}}$	$\mathbf{1.52 \cdot 10^{-2}}$	$\mathbf{2.90 \cdot 10^{-4}}$	$\mathbf{1.05 \cdot 10^{-2}}$
CombMP	$5.81 \cdot 10^{-4}$	$1.12 \cdot 10^{-2}$	$3.76 \cdot 10^{-4}$	$1.06 \cdot 10^{-2}$	$1.07 \cdot 10^{-3}$	$1.88 \cdot 10^{-2}$	$7.04 \cdot 10^{-4}$	$1.61 \cdot 10^{-2}$
SetMP (Gen.)	–	–	–	–	$8.50 \cdot 10^{-4}$	$1.63 \cdot 10^{-2}$	$5.26 \cdot 10^{-4}$	$1.26 \cdot 10^{-2}$
zfp	$9.19 \cdot 10^{-5}$	$7.34 \cdot 10^{-3}$	$4.02 \cdot 10^{-7}$	$4.84 \cdot 10^{-4}$	$1.04 \cdot 10^{-4}$	$7.83 \cdot 10^{-3}$	$4.18 \cdot 10^{-7}$	$4.95 \cdot 10^{-4}$

Training and Validation Pipeline

All models are trained for a maximum of 200 epochs in Abilene dataset and 500 in Geant using Adam optimizer (with learning rate 0.003 and epsilon 0.001) and using batches of 25 samples if required. The model state corresponding to the best performing iteration over the validation set is selected for the test evaluation.

6.3.2 Experimental Results

Table 6.1 shows the reconstruction error (MSE and MAE) obatained by our framework and the baselines in both datasets for two different compression factors (1/3 and 2/3). We can see that SetMP, our topological edge-less architecture, clearly outperforms the other ML methods in all scenarios –improving on average by 75% and 48%, respectively, the best MSE and MAE obtained by ML baselines–, followed by our most generic CombMP architecture. Focusing only on the ML baselines, the customised MPNN is the best performing solution, followed by the MLP and with the standard GNNs in the last place.

Table 6.1 also shows the performance of the state-of-the-art lossy compression method *zfp* for the desired precision. As it can be seen, *zfp* gets better reconstruction errors than our model in all scenarios, although we note that differences are considerably lower for smaller (i.e. more challenging) compression factors. Overall, we reckon that these are promising results; our TDL-based models are postulated as strong ML-based baselines for network traffic compression, and by addressing some of their current limitations (see next Section 6.5) there can be room for shortening the gap with respect to the state-of-the-art.

Finally, we also performed an exploratory evaluation of the generalization capabilities of our proposed topological-inspired architectures. To do so, we took the best performing SetMP models trained on Abilene and evaluated them in the Geant network; results are shown in Table 6.1, labelled by "SetMP (Gen.)". Despite the slight decrease of performance w.r.t. the SetMP models trained on Geant, these models obtain better reconstruction errors than all other ML architectures, including CombMP.

6.4 Related Work

There already exist ML-based models in the literature that target lossy compression tasks [152, 153], but they are mainly entangled to Information Theory concepts used in classical compression algorithms, and applied to Computer Vision domains. Our approach differs from them in these two basic aspects, and it is aligned with the pipeline of *zfp* [147], the state-of-the-art lossy method for floating-point data compression. As detailed in [147], the first step of *zfp* consists in dividing floating matrices or tensors in disjoint blocks of a fixed dimension, which then are independently processed to extract compressed representations. This has obvious similarities with our search of higher-order structures, with the difference that in *zfp* the divisions are totally determined by the input elements' order.

Precisely our proposed topology inference module has some resemblances to that of [154], which is also based on grouping entries of an affinity (i.e. similarity) matrix computed over a set of element's neural embeddings. Nevertheless, due to the constraints imposed by the compression task there exists relevant differences between that inference procedure and ours: whereas we design a clustering methodology to get disjoint and uniform-length sets, in [154] they implement a multi-scale hyperedge forming pipeline where each node ends up belonging to an arbitrary number of higher-order structures –and not only among different hyperedge degrees, but also within the same scale.

Regarding our topological-inspired MP architectures, we highlight the paper of [31] on Combinatorial Complexes (CCC) and the survey [155] on TDL architectures. Our most general method –CombMP– was conceptualized before the publication of these works, but we note that it can be formalized in terms of CCCs' notation by considering nodes as 0-cells, edges as 1-cells, and hyperedges as 2-cells; it is due to this fact that we called it CombMP, which stands for *Combinatorial Message Passing*. SetMP, on the other hand, belongs to the hypergraph family of TDL architectures according to the classification of [155]. More precisely, it can be linked to DeepSets architectures [156], which is why it received that name.

Finally, a special mention should be given to [157], which also leverages a ML model to specifically perform traffic compression on networks with multiple vantage points. However, authors of this work implement a spatio-temporal GNN that acts as a *predictor* of a traditional lossless compression method (in this case, Arithmetic Coding), which defines a totally different conceptual approach.

6.5 Discussion

The obtained results shown in Section 6.3 support our hypothesis that taking into account higher-order interactions could help in designing more expressive ML-based models for (graph) signal compression tasks, specially due to the fact that these higher-order structures can go beyond the (graph) local neighborhood and connect possibly distant datapoints whose signals may be strongly correlated (e.g. between generator and sink nodes). In that regard, TDL can provide us with novel methodologies that naturally encompass and exploits those multi-element relations. Moreover, it is interesting to see how our set-based architecture outperforms the combinatorial-based one in every scenario, suggesting that intermediate binary connections might add noise in the process of distilling compressed representations.

However, despite the promising preliminary results, and because of them as well, there are many aspects and limitations of our proposed methodology that are being investigated and will be addressed in future work. The following list summarizes some of the main lines of research:

Topology Inference We hypothesize that the main limitation of our current method revolves around this module, as it can potentially gather elements that do not necessarily share any correlation –especially at the end of the clustering process, where hyperedges are built in a greedy manner regardless of the actual similarities between their elements. Whereas *zfp* implements a sophisticated method to deal with such uncorrelated elements, our current methodology mainly relies on existing correlations to perform compression; this can lead to a systematic poor reconstruction error of the involved signals. In order to address this, apart from exploring other metrics beyond SNR, it would be interesting to consider more flexible clustering approaches that could dynamically adapt the length/number of the inferred higher-order structures (e.g. by defining a Reinforcement Learning pipeline to perform the division, or adapting a solution like the Differentiable Cell Complex Module [158] to our scenario).

Implementation Issues Our current proposal does not scale well with the original (graph) signal dimension, as it mainly relies on an iterative pair-wise similarity computation between all (graph) elements. More research about how to relax this aspect is required in order to make its deployment feasible, and in this regard some ideas we want to explore are:

- To test whether static higher-order structures generated from training samples can perform well in testing time; not only this would reduce the execution time, but also the necessity of storing the cluster sequence at each iteration.
- To check the feasibility of storing simultaneously sparse matrices indicating when the compression loss exceeds a certain threshold; apart from becoming a potential

indicator of anomalies in the signal, if the distribution of big reconstruction errors is really sparse, combining them with the compressed representations will provide with loss tolerance guarantees, and could be key for matching *zfp* performance.

- For large graphs, to divide the original graph into independent subgraphs, to which then our method is applied.

Topological MP Another goal is to shed more light on the performance difference observed between our two architectures, SetMP and CombMP. Owing to current results, it seems that the lower complexity of SetMP is a clear advantage, and suggests that intermediate edge communications noisily interfere in the compression process. This should be further validated with other datasets, and possibly by testing some variations of the edge-based communication pipeline in the general CombMP architecture.

Chapter 7

Conclusions and Future Work

This dissertation addressed the challenges of optimizing complex systems that naturally emerge in our digital era, specially due to the explosive growth of networked applications and connected devices derived from technological advances over the last decades. In this context, we aimed at investigating novel Machine Learning (ML) based methodologies able to exploit the distributed nature of such networked environments to address the scalability issues of traditional solutions.

Our investigation lead us to the design of MAGNNETO, a general framework for distributed optimization in networked systems. Through the combination of Multi-Agent Reinforcement Learning (MARL) and Graph Neural Network (GNN) technologies, the modular-based MAGNNETO architecture is able to naturally distribute the optimization among a set of agents deployed across a networked scenario. Thanks to the implemented message-passing scheme from the GNN-based modelling of the environment, these agents can effectively cooperate to achieve a global optimization goal by leveraging low-intensive communications between neighbors, which in turn can be propagated through the real network links.

This dissertation also explored the applicability of MAGNNETO to different real-world scenarios. The MARL setting implemented by MAGNNETO can be adapted to accommodate different distributed systems by the definition of the environment (which elements of the network the agents represent, and how they are interconnected), the state and action spaces (based on the features of the network and possible changes that agents may execute), as well as the reward function (which provides feedback about the performed actions). In particular, we visited in depth two computer network use cases: intradomain Traffic Engineering (TE) in Internet Service Provider (ISP) networks, and Congestion Control (CC) optimization in Datacenter Networks (DCNs).

Regarding intradomain TE, we introduce the adaptation MAGNNETO-TE and demonstrate that it represents a significant leap forward in this domain. MAGNNETO-TE not only matches the near-optimal performance exhibited by current state-of-the-art TE optimizers, but surpasses them considerably in terms of execution speed, reacting much quicker to traffic changes without a noticeable decrease in performance. This system notably thrives in scenarios unseen during the training phase, showcasing an impressive generalization power.

In a similar vein, the MAGNNETO-CC adaptation offers a promising approach towards dynamic ECN tuning within CC in DCNs. Outperforming previous baselines in flow completion time and maintaining shorter queue lengths, it addresses the necessities of next-generation DCNs adeptly. The solution's remarkable generalization capabilities allow for more robust and adaptable agent implementations, capable of accommodating significant variations in network scenarios compared to the conditions experienced during training.

Overall, MAGNNETO framework displays notable versatility, adapting well to these two use cases in the computer network sector due to its distributed nature. In both scenarios, it shows an excellent balance between performance and execution cost, unprecedented generalization capabilities, and fully compatibility with current and legacy hardware. We argue that these are relevant features towards a feasible deployment and commercialization, a capability that has been largely overlooked by previous ML-based solutions.

In conclusion, the MAGNNETO architecture stands as a beacon of innovation and adaptability in the evolving landscape of network optimization, offering solutions that are not only highly efficient and expedited but also marked by their strong generalization abilities. Its applicability, extending potentially beyond the networking domain, underscores its position as a potent tool in addressing contemporary challenges in networked systems, ushering in a new era of optimized, responsive, and adaptive network operations. Future explorations and adaptations of the MAGNNETO framework are eagerly anticipated, potentially heralding revolutionary applications in diverse fields.

7.1 Future Work

As proposed in previous Chapter 6, we reckon that the exploration of Topological Deep Learning (TDL) methodologies [30, 31] to boost MAGNNETO beyond the graph domain is a promising research towards the operation of very complex dynamical distributed systems. Our preliminary results suggest that topological-inspired methodologies can potentially exploit multi-way correlations and long range interactions in networked scenarios, hence addressing some of the main limitations of current GNN models. Moreover, since TNNs share the same modular-based architecture than their graph counterparts, a topological-enhanced MAGNNETO framework could also capitalize on the inherent distributed nature of networked systems and naturally parallellize the optimization process through them.

We also note that, through the definition of the reward function in the Reinforcement Learning (RL) setting, MAGNNETO is trained based on a static, pre-defined optimization goal. In this regard, another interesting line of research would be to explore the use of meta-learning techniques [159] in order to design an objective-agnostic optimization framework, able to dynamically adapt its optimizations goals to perform different tasks without the need of re-training it. This would be aligned to the trend of intent-based networking, which aims to operate networks by means of a declarative language that expresses what the network administrator wants the network to do –as opposed to using traditional imperative languages that express how the network must run.

Finally, we argue that MAGNNETO can be considered to optimize other challenging and impactful scenarios apart from the two relevant use cases considered in this dissertation. For instance, and still within the field of computer networks, one challenging but very interesting possibility would be to extend MAGNNETO to handle wireless networks (e.g. 5G/6G mobile networks), where the network topology can dynamically change over time. However, we recall that MAGNNETO was conceived to optimize networked systems in general, so future research might also consider MAGNNETO in other scenarios beyond the scope of computer networks –such as traffic lights in smart cities, autonomous vehicles, etc.

On this matter, we highlight that MAGNNETO has been already successfully adaptated to tackle the optimal power flow problem in electrical power systems [19], where the goal is to find the best operating point by optimizing the power output of generators in power grids. After a careful adaptation of MAGNNETO to this scenario, it has shown great generalization capabilities as well as a significant cost reduction (up to 30%) w.r.t. standard solutions on that field; we refer the reader to [58] and [160] works for all the details of this application. We believe that this fact showcases the potential of MAGNNETO's architecture towards optimizing distributed systems in a broader sense.

Appendices

Appendix A

Outcomes of the Thesis

A.1 Related Publications

Conference Papers:

- 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.
  - Oral presentation at the Main Conference program.
  - Runner-Up for the Best Paper Award.

- G. Bernárdez, L. Telyatnikov, E. Alarcón, A. Cabellos-Aparicio, P. Barlet-Ros, and P. Liò, "Topological Network Traffic Compression", in Proceedings of the 2nd Graph Neural Networking Workshop 2023 (GNNet), ACM CoNEXT 2023.
  - Oral presentation at the Workshop program.

- G. Bernárdez, L. Telyatnikov, E. Alarcón, A. Cabellos-Aparicio, P. Barlet-Ros, and P. Liò, "Topological Graph Signal Compression". Extended Abstract accepted at the 2nd Learning on Graphs Conference 2023 (LoG).
  - Oral presentation at the Main Conference program.

Journal Papers:

- G. Bernárdez, J. Suárez-Varela, A. López, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "MAGNNETO: A graph neural network-based multi-agent system for traffic engineering," IEEE Transactions on Cognitive Communications and Networking **(JCR Q1)**, vol. 9, no. 2, pp. 494–506, 2023.

- **(under review)** G. Bernárdez, J. Suárez-Varela, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "GraphCC: A practical Graph Learning-based Approach to Congestion Control in Datacenters". Submitted to IEEE Transactions on Cognitive Communications and Networking **(JCR Q1)**.

A.2 Related Patents

- **(published)** G. Bernárdez, J. Suárez-Varela, M. Ferriol, B. Wu, S. Xiao, X. Cheng, L. Wenjie, L. Fenglin, P. Barlet-Ros, and A. Cabellos-Aparicio. "Devices and methods for autonomous distributed control of computer networks". Huawei Technologies Co., LTD.
  - International PCT Application Number PCT/CN2021/091915, filed in May 6th 2021.
  - Patent Publication Number WO/2022/232994, published in November 10th 2022.

- **(filed)** G. Bernárdez, J. Suárez-Varela, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Device and Method for an Agent for Dynamically Adapting an Explicit Congestion Notification Configuration in a Network System". Huawei Technologies Co., LTD.
  - International PCT Application Number PCT/CN2022/138121, filed in December 9th 2022.

A.3 Other Merits

- 3 × Winner of ICML 2023 Topological Deep Learning Challenge,[1] in team with L. Scofano, I. Spinelli, S. Scardapane, S. Fiorellino, O. Zaghen, L. Telyatninkov and C. Battiloro. We participated in 3 out of the 4 different categories proposed in the challenge, and obtained the first position in all three with the following submissions:
  - In the Simplicial Domain for our Simplicial Attention Network implementation.
  - In the Cellular Domain for our Cell Attention Network implementation.
  - In the Hypergraph Domain for our AllSetTransformer implementation.

- 6-month research stay at the University of Cambridge under the supervision of Prof. Pietro Liò, from 2023-01-01 to 2023-06-30. The main research focused on the exploration and design of novel Topological and Geometric Deep Learning methodologies to compress signals over graphs.

- Accepted Poster at the 6th Edition of the Graph Signal Processing Workshop 2023 (June 12-14 in Oxford, UK), presenting our journal publication "MAGNNETO: A graph neural network-based multi-agent system for traffic engineering".

- Doctoral fellowship from the Secretariat for Universities and Research of the Ministry of Business and Knowledge of the Government of Catalonia (ref. 2020 FISDU 00416), FI SDUR grant October 2020.

[1] `https://pyt-team.github.io/topomodelx/challenge`

A.4 Other Publications

- J. Suárez-Varela, M. Ferriol-Galmés, A. López, P. Almasan, G. Bernárdez, D. Pujol-Perich, K. Rusek, L. Bonniot, C. Neumann, F. Schnitzler, F. Taïani, M. Happ, C. Maier, J. Lei Du, M. Herlich, P. Dorfinger, N. Vincent Hainke, S. Venz, J. Wegener, H. Wissing, B. Wu, S. Xiao, P. Barlet-Ros, and A. 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
- Á. López-Cardona, G. Bernárdez, P. Barlet-Ros, and A. Cabellos-Aparicio, "Proximal policy optimization with graph neural networks for optimal power flow," arXiv preprint arXiv:2212.12470, 2022.
- **(under review)** L. Telyatnikov, M.S. Bucarelli, G. Bernárdez, O. Zaghen, S. Scardapane and P. Liò, "Hypergraph Neural Networks through the Lens of Message Passing: A Common Perspective to Homophily and Architecture Design", submitted to the 12th International Conference on Learning Representations (ICLR) 2024.
- Co-author of "ICML 2023 Topological Deep Learning Challenge: Design and Results", in Topological, Algebraic and Geometric Learning Workshops 2023. PMLR, 2023. p. 3-8.
- **(under review)** Co-author of "TopoX: A Suite of Python Packages for Machine Learning on Topological Domains", submitted to the Journal of Machine Learning Research.

Bibliography

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] "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/

[8] 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.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[11] 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.

[12] K. Sharifani and M. Amini, “Machine learning and deep learning: A review of methods and applications,” *World Information Technology and Engineering Journal*, vol. 10, no. 07, pp. 3897–3904, 2023.

[13] S. Tosserams, L. P. Etman, and J. Rooda, “A classification of methods for distributed system optimization based on formulation structure,” *Structural and Multidisciplinary Optimization*, vol. 39, pp. 503–517, 2009.

[14] W. Ahmed and Y. W. Wu, “A survey on reliability in distributed systems,” *Journal of Computer and System Sciences*, vol. 79, no. 8, pp. 1243–1255, 2013.

[15] R. Bhatia, F. Hao, M. Kodialam, and T. Lakshman, “Optimized network traffic engineering using segment routing,” in *2015 IEEE Conference on Computer Communications (INFOCOM)*, 2015, pp. 657–665.

[16] B. Fortz and M. Thorup, “Internet traffic engineering by optimizing ospf weights,” in *Proceedings of IEEE INFOCOM*, vol. 2, 2000, pp. 519–528.

[17] S. Gay, R. Hartert, and S. Vissicchio, “Expect the unexpected: Sub-second optimization for segment routing,” in *IEEE INFOCOM*, 2017, pp. 1–9.

[18] 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,” *ACM SIGCOMM computer communication review*, vol. 45, no. 4, pp. 15–28, 2015.

[19] A. Wood and B. Wollenberg, *Power generation, operation, and control*, 01 2012.

[20] J. Baillieul and P. J. Antsaklis, “Control and communication challenges in networked real-time systems,” *Proceedings of the IEEE*, vol. 95, no. 1, pp. 9–28, 2007.

[21] J. N. Foerster, “Deep multi-agent reinforcement learning,” Ph.D. dissertation, University of Oxford, 2018.

[22] R. Ding, Y. Yang, J. Liu, H. Li, and F. Gao, “Packet routing against network congestion: A deep multi-agent reinforcement learning approach,” in *2020 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2020, pp. 932–937.

[23] N. Geng, T. Lan, V. Aggarwal, Y. Yang, and M. Xu, “A multi-agent reinforcement learning perspective on distributed traffic engineering,” in *IEEE International Conference on Network Protocols (ICNP)*, 2020, pp. 1–11.

[24] S. Yan, X. Wang, X. Zheng, Y. Xia, D. Liu, and W. Deng, “ACC: Automatic ECN tuning for high-speed datacenter networks,” in *ACM SIGCOMM*, 2021.

[25] T. Chu, S. Chinchali, and S. Katti, “Multi-agent reinforcement learning for networked system control,” in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=Syx7A3NFvH

[26] 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.

[27] J. Jiang, C. Dun, T. Huang, and Z. Lu, “Graph convolutional reinforcement learning,” in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HkxdQkSYDB

[28] J. Su, S. Adams, and P. A. Beling, “Counterfactual multi-agent reinforcement learning with graph convolution communication,” *arXiv preprint arXiv:2004.00470*, 2020.

[29] A. K.-L. Mok, “Fundamental design problems of distributed systems for the hard-real-time environment,” Ph.D. dissertation, Massachusetts Institute of Technology, 1983.

[30] C. Bodnar, “Topological deep learning: Graphs, complexes, sheaves,” Ph.D. dissertation, Apollo - University of Cambridge Repository, 2022. [Online]. Available: https://www.repository.cam.ac.uk/handle/1810/350982

[31] M. Hajij, G. Zamzmi, T. Papamarkou, N. Miolane, A. Guzmán-Sáenz, K. N. Ramamurthy, T. Birdal, T. K. Dey, S. Mukherjee, S. N. Samaga, N. Livesay, R. Walters, P. Rosen, and M. T. Schaub, “Topological deep learning: Going beyond graph data,” 2023.

[32] 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.

[33] G. Bernárdez, J. Suárez-Varela, A. López, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, “Magnneto: A graph neural network-based multi-agent system for traffic engineering,” *IEEE Transactions on Cognitive Communications and Networking*, vol. 9, no. 2, pp. 494–506, 2023.

[34] G. Bernárdez, J. Suárez-Varela, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, “Graphcc: A practical graph learning-based approach to congestion control in datacenters,” *arXiv preprint arXiv:2308.04905*, 2023.

[35] G. Bernárdez, J. Suárez-Varela, A. López, B. Wu, S. Xiao, X. Cheng, W. Liu, S. Xu, P. Barlet-Ros, and A. Cabellos-Aparicio, “Devices and methods for autonomous distributed control of computer networks,” Patent Publication Number WO/2022/232994, published on Nov. 10 2022.

[36] G. Bernárdez, J. Suárez-Varela, X. Shi, S. Xiao, X. Cheng, P. Barlet-Ros, and A. Cabellos-Aparicio, “Device and method for an agent for dynamically adapting an explicit congestion notification configuration in a network system,” Patent Application Number PCT/CN2022/138121, international filing on Dec. 9 2022.

[37] G. Bernárdez, L. Telyatnikov, E. Alarcón, A. Cabellos-Aparicio, P. Barlet-Ros, and P. Liò, “Topological graph signal compression,” *arXiv preprint arXiv:2308.11068*, 2023.

[38] G. Bernárdez, L. Telyatnikov, E. Alarcón, A. Cabellos-Aparicio, P. Barlet-Ros, and P. Lio, “Topological network traffic compression,” 2023.

[39] 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.

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

[41] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proceedings of Advances in neural information processing systems (NIPS)*, 2016, pp. 3844–3852.

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

[43] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[44] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?" in *International Conference on Learning Representations*, 2021.

[45] 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.

[46] P. Veličković, "Message passing all the way up," in *ICLR 2022 Workshop on Geometrical and Topological Representation Learning*, 2022.

[47] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

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

[49] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming.* Athena Scientific Belmont, MA, 1996, vol. 5.

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

[51] 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.

[52] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[53] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

[55] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning.* PMLR, 2015, pp. 1889–1897.

[56] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[57] F. A. Oliehoek, M. T. J. Spaan, and N. Vlassis, "Optimal and approximate q-value functions for decentralized pomdps," *J. Artif. Int. Res.*, vol. 32, no. 1, pp. 289–353, May 2008.

[58] Á. López-Cardona, G. Bernárdez, P. Barlet-Ros, and A. Cabellos-Aparicio, "Proximal policy optimization with graph neural networks for optimal power flow," *arXiv preprint arXiv:2212.12470*, 2022.

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

[60] 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.

[61] 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.

[62] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, pp. 1–99, 2018.

[63] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, "Requirements for traffic engineering over MPLS," Internet Requests for Comments, RFC 2702, 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2702.txt

[64] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, "Overview and principles of internet traffic engineering," Internet Requests for Comments, RFC 3272, 2002. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3272.txt

[65] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke, "Optimal oblivious routing in polynomial time," *Journal of Computer and System Sciences*, vol. 69, no. 3, pp. 383–394, 2004.

[66] B. Fortz and M. Thorup, "Increasing internet capacity using local search," *Computational Optimization and Applications*, vol. 29, no. 1, pp. 13–48, 2004.

[67] J. Moy, "Ospf version 2," Internet Requests for Comments, RFC Editor, STD 54, April 1998. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2328.txt

[68] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 36–56, 2008.

[69] A. Mendiola, J. Astorga, E. Jacob, and M. Higuero, "A survey on the contributions of software-defined networking to traffic engineering," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 918–953, 2016.

[70] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, "Learning to route," in *Proceedings of the ACM Workshop on Hot Topics in Networks (HotNets)*, 2017, pp. 185–191.

[71] 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 Conference on Computer Communications (INFOCOM)*, 2018, pp. 1871–1879.

[72] G. Stampa, M. Arias, D. Sanchez-Charles, V. Muntés-Mulero, and A. Cabellos, "A deep-reinforcement learning approach for software-defined networking routing optimization," *CoNEXT Student Workshop*, 2017.

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

[74] 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.

[75] C. Filsfils, N. K. Nainar, C. Pignataro, J. C. Cardona, and P. Francois, “The segment routing architecture,” in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[76] X. Huang, S. Cheng, K. Cao, P. Cong, T. Wei, and S. Hu, “A survey of deployment solutions and optimization strategies for hybrid sdn networks,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1483–1507, 2018.

[77] Cisco, “Planning and designing networks with the cisco mate portfolio,” white paper, 2013.

[78] Juniper, “WANDL IP/MPLSView,” http://www.juniper.net/assets/us/en/local/pdf/datasheets/1000500-en.pdf.

[79] A. Sridharan, R. Guerin, and C. Diot, “Achieving near-optimal traffic engineering solutions for current ospf/is-is networks,” *IEEE/ACM Transactions On Networking*, vol. 13, no. 2, pp. 234–247, 2005.

[80] I. Minei and J. Lucek, *MPLS-Enabled Applications.* John Wiley & Sons, Ltd, 2005.

[81] D. Xu, M. Chiang, and J. Rexford, “Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering,” *IEEE/ACM Transactions on networking*, vol. 19, no. 6, pp. 1717–1730, 2011.

[82] T. Benson, A. Anand, A. Akella, and M. Zhang, “Microte: Fine grained traffic engineering for data centers,” in *Proceedings of the Seventh COnference on emerging Networking EXperiments and Technologies*, 2011, pp. 1–12.

[83] M. Luo, “Dsox: Tech report,” Technical Report, Huawei Shannon Lab, Tech. Rep., 2013.

[84] A. Basu and J. Riecke, “Stability issues in ospf routing,” *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 225–236, 2001.

[85] K. Rusek, J. Suárez-Varela, A. Mestres, P. Barlet-Ros, and A. Cabellos-Aparicio, “Unveiling the potential of graph neural networks for network modeling and optimization in sdn,” in *Proceedings of the ACM Symposium on SDN Research (SOSR)*, 2019, pp. 140–151.

[86] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem, “What matters for on-policy deep actor-critic methods? a large-scale study,” in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=nIAxjsniDzg

[87] M. Roughan, “Simplifying the synthesis of internet traffic matrices,” *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 93–96, 2005.

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

[89] 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.

[90] J. Suárez-Varela, S. Carol-Bosch, K. Rusek, P. Almasan, M. Arias, P. Barlet-Ros, and A. Cabellos-Aparicio, "Challenging the generalization capabilities of graph neural networks for network modeling," in *Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*. ACM, 2019, pp. 114–115.

[91] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[92] 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?" *IEEE International Conference on Network Protocols (ICNP)*, Nov. 2021.

[93] J. Suárez-Varela, A. Mestres, J. Yu, L. Kuang, H. Feng, P. Barlet-Ros, and A. Cabellos-Aparicio, "Feature engineering for deep reinforcement learning based routing," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[94] H. Mao, W. Liu, J. Hao, J. Luo, D. Li, Z. Zhang, J. Wang, and Z. Xiao, "Neighborhood cognition consistent multi-agent reinforcement learning," *arXiv preprint arXiv:1912.01160*, 2019.

[95] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2. IEEE, 2005, pp. 729–734.

[96] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio, "Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case," *arXiv preprint arXiv:1910.07421*, 2019.

[97] 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.

[98] W. Li, J. Liu, S. Wang, T. Zhang, S. Zou, J. Hu, W. Jiang, and J. Huang, "Survey on traffic management in data center network: From link layer to application layer," *IEEE Access*, vol. 9, pp. 38 427–38 456, 2021.

[99] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *ACM SIGCOMM*, 2010.

[100] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale RDMA deployments," *ACM SIGCOMM*, 2015.

[101] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM*, 2015.

[102] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *ACM SIGCOMM*, 2018, pp. 221–235.

[103] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh *et al.*, “HPCC: High precision congestion control,” in *ACM SIGCOMM*, 2019, pp. 44–58.

[104] G. Kumar, N. Dukkipati, K. Jang, H. M. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan *et al.*, “Swift: Delay is simple and effective for congestion control in the datacenter,” 2020, pp. 514–528.

[105] A. Saeed, V. Gupta, P. Goyal, M. Sharif, R. Pan, M. Ammar, E. Zegura, K. Jang, M. Alizadeh, A. Kabbani *et al.*, “Annulus: A dual congestion control loop for datacenter and wan traffic aggregates,” in *ACM SIGCOMM*, 2020, pp. 735–749.

[106] K. Ramakrishnan, S. Floyd, and D. Black, “RFC3168: The addition of explicit congestion notification (ECN) to IP,” 2001.

[107] D. Shan and F. Ren, “ECN marking with micro-burst traffic: Problem, analysis, and improvement,” *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1533–1546, 2018.

[108] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, “WCMP: Weighted cost multipathing for improved fairness in data centers,” in *European Conference on Computer Systems*, 2014, pp. 1–14.

[109] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, “F10: A fault-tolerant engineered network,” in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2013, pp. 399–412.

[110] P. Gill, N. Jain, and N. Nagappan, “Understanding network failures in data centers: measurement, analysis, and implications,” in *ACM SIGCOMM*, 2011, pp. 350–361.

[111] L. Chen, J. Lingys, K. Chen, and F. Liu, “AuTO: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization,” in *ACM SIGCOMM*, 2018.

[112] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, “A deep reinforcement learning perspective on internet congestion control,” in *International Conference on Machine Learning*, 2019, pp. 3050–3059.

[113] S. Abbasloo, C.-Y. Yen, and H. J. Chao, “Classic meets modern: A pragmatic learning-based congestion control for the internet,” in *ACM SIGCOMM*, 2020.

[114] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, “pFabric: Minimal near-optimal datacenter transport,” *ACM SIGCOMM*, 2013.

[115] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, “Information-agnostic flow scheduling for commodity data centers,” in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015, pp. 455–468.

[116] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, “pHost: Distributed near-optimal datacenter transport over commodity network fabric,” in *ACM CoNEXT*, 2015.

[117] I. Cho, K. Jang, and D. Han, “Credit-scheduled delay-bounded congestion control for datacenters,” in *ACM SIGCOMM*, 2017, pp. 239–252.

[118] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *ACM SIGCOMM*, 2020.

[119] J. Zhang, W. Bai, and K. Chen, "Enabling ECN for datacenter networks with RTT variations," in *ACM CoNEXT*, 2019, pp. 233–245.

[120] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting network support for rdma," in *ACM SIGCOMM*, 2018, pp. 313–326.

[121] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li, "In-band network telemetry: a survey," *Computer Networks*, vol. 186, p. 107763, 2021.

[122] "In-situ operations, administration, and maintenance (iOAM)," https://github.com/CiscoDevNet/iOAM, Accessed: 2022-01-25.

[123] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on networking*, vol. 1, no. 4, pp. 397–413, 1993.

[124] Broadcom, "Broadcom StrataXGS switch solutions," https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs, 2022, accessed: 2022-28-01.

[125] ——, "Broadcom product news releases," https://www.broadcom.com/company/news/product-releases, 2022, accessed: 2022-28-01.

[126] Intel, "Intel tofino series programmable ethernet switch asic," https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html, 2022, accessed: 2022-28-01.

[127] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 123–137.

[128] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, 2006.

[129] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav *et al.*, "Conga: Distributed congestion-aware load balancing for datacenters," in *ACM SIGCOMM*, 2014, pp. 503–514.

[130] P. Goyal, P. Shah, K. Zhao, G. Nikolaidis, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2022, pp. 779–805.

[131] L. Ruiz, L. Chamon, and A. Ribeiro, "Graph neural networks and the transferability of graph neural networks," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[132] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM*, 2015.

[133] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*, 2010, pp. 15–34.

[134] PFC, "IEEE 802.11Qbb. priority based flow control," 2021, 2011.

[135] B. Stephens, A. L. Cox, A. Singla, J. Carter, C. Dixon, and W. Felter, "Practical DCB for improved data center networks," in *IEEE INFOCOM*, 2014.

[136] J. H. Giraldo, F. D. Malliaros, and T. Bouwmans, "Understanding the relationship between over-smoothing and over-squashing in graph neural networks," *arXiv preprint arXiv:2212.02374*, 2022.

[137] K. Nguyen, N. M. Hieu, V. D. Nguyen, N. Ho, S. Osher, and T. M. Nguyen, "Revisiting over-smoothing and over-squashing using ollivier-ricci curvature," in *International Conference on Machine Learning*. PMLR, 2023, pp. 25 956–25 979.

[138] C. Bodnar, F. Frasca, Y. Wang, N. Otter, G. F. Montufar, P. Lio, and M. Bronstein, "Weisfeiler and lehman go topological: Message passing simplicial networks," in *International Conference on Machine Learning*. PMLR, 2021, pp. 1026–1037.

[139] C. Bodnar, F. Frasca, N. Otter, Y. Wang, P. Lio, G. F. Montufar, and M. Bronstein, "Weisfeiler and lehman go cellular: Cw networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2625–2640, 2021.

[140] F. Battiston, E. Amico, A. Barrat, G. Bianconi, G. Ferraz de Arruda, B. Franceschiello, I. Iacopini, S. Kéfi, V. Latora, Y. Moreno *et al.*, "The physics of higher-order interactions in complex systems," *Nature Physics*, vol. 17, no. 10, pp. 1093–1098, 2021.

[141] S. Barbarossa and S. Sardellitti, "Topological signal processing over simplicial complexes," *IEEE Transactions on Signal Processing*, vol. 68, pp. 2992–3007, 2020.

[142] P. S. Skardal, L. Arola-Fernández, D. Taylor, and A. Arenas, "Higher-order interactions improve optimal collective dynamics on networks," *arXiv preprint arXiv:2108.08190*, 2021.

[143] M. T. Schaub, A. R. Benson, P. Horn, G. Lippner, and A. Jadbabaie, "Random walks on simplicial complexes and the normalized hodge 1-laplacian," *SIAM Review*, vol. 62, no. 2, pp. 353–391, 2020.

[144] Y. Schiff, V. Chenthamarakshan, K. N. Ramamurthy, and P. Das, "Characterizing the latent space of molecular deep generative models with persistent homology metrics," *arXiv preprint arXiv:2010.08548*, 2020.

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

[146] 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, 2016.

[147] J. Diffenderfer, A. L. Fox, J. A. Hittinger, G. Sanders, and P. G. Lindstrom, "Error analysis of zfp compression for floating-point data," *SIAM Journal on Scientific Computing*, vol. 41, no. 3, pp. A1867–A1898, 2019.

[148] 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.

[149] T. Yuan, W. Deng, J. Tang, Y. Tang, and B. Chen, "Signal-to-noise ratio: A robust distance metric for deep metric learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4815–4824.

[150] T. N. Kipf and M. Welling, “Semi-Supervised Classification with Graph Convolutional Networks,” in *ICLR*, 2017.

[151] M. Fey and J. E. Lenssen, “Fast graph representation learning with pytorch geometric,” *arXiv preprint arXiv:1903.02428*, 2019.

[152] M. Havasi, “Advances in compression using probabilistic models,” 2021. [Online]. Available: https://www.repository.cam.ac.uk/handle/1810/331555

[153] Y. Yang, S. Mandt, L. Theis *et al.*, “An introduction to neural data compression,” *Foundations and Trends® in Computer Graphics and Vision*, vol. 15, no. 2, pp. 113–200, 2023.

[154] C. Xu, M. Li, Z. Ni, Y. Zhang, and S. Chen, “Groupnet: Multiscale hypergraph neural networks for trajectory prediction with relational reasoning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 6498–6507.

[155] M. Papillon, S. Sanborn, M. Hajij, and N. Miolane, “Architectures of topological deep learning: A survey on topological neural networks,” *arXiv preprint arXiv:2304.10031*, 2023.

[156] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, “Deep sets,” *Advances in neural information processing systems*, vol. 30, 2017.

[157] P. Almasan, K. Rusek, S. Xiao, X. Shi, X. Cheng, A. Cabellos-Aparicio, and P. Barlet-Ros, “Leveraging spatial and temporal correlations for network traffic compression,” *arXiv preprint arXiv:2301.08962*, 2023.

[158] C. Battiloro, I. Spinelli, L. Telyatnikov, M. Bronstein, S. Scardapane, and P. Di Lorenzo, “From latent graph to latent topology inference: Differentiable cell complex module,” *arXiv preprint arXiv:2305.16174*, 2023.

[159] Y. Tian, X. Zhao, and W. Huang, “Meta-learning approaches for learning-to-learn in deep learning: A survey,” *Neurocomputing*, vol. 494, pp. 203–223, 2022.

[160] Á. Lopez Cardona, “Optimal power flow computation using neural networks,” Master’s thesis, Universitat Politècnica de Catalunya, 2022.