MAGNNETO: A Graph Neural Network-based Multi-Agent system for Traffic Engineering

Guillermo Bernárdez, José Suárez-Varela, Albert López, Xiang Shi, Shihan Xiao, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio

***Abstract*— Current trends in networking propose the use of Machine Learning (ML) for a wide variety of network optimization tasks. As such, many efforts have been made to produce ML-based solutions for Traffic Engineering (TE), which is a fundamental problem in ISP networks. Nowadays, state-of-the-art TE optimizers rely on traditional optimization techniques, such as Local search, Constraint Programming, or Linear programming. In this paper, we present MAGNNETO, a distributed ML-based framework that leverages Multi-Agent Reinforcement Learning and Graph Neural Networks for distributed TE optimization. MAGNNETO deploys a set of agents across the network that learn and communicate in a distributed fashion via message exchanges between neighboring agents. Particularly, we apply this framework to optimize link weights in OSPF, with the goal of minimizing network congestion. In our evaluation, we compare MAGNNETO against several state-of-the-art TE optimizers in more than 75 topologies (up to 153 nodes and 354 links), including realistic traffic loads. Our experimental results show that, thanks to its distributed nature, MAGNNETO achieves comparable performance to state-of-the-art TE optimizers with significantly lower execution times. Moreover, our ML-based solution demonstrates a strong generalization capability to successfully operate in new networks unseen during training.**

***Index Terms*—Traffic Engineering, Routing Optimization, Multi-Agent Reinforcement Learning, Graph Neural Networks**

I. Introduction

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* [1], [2]. In this vein, Machine Learning (ML) is considered as a promising technique for producing efficient tools for autonomous networking [3], [4].

In this paper, we revisit a fundamental networking problem: Traffic Engineering (TE) optimization [5], [6]. TE is among the most common operation tasks in today's 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 [7]–[11]. 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., OSPF [12]), making fine-grained flow-based routing, or re-routing traffic across overlay networks [13], [14].

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., [15]–[18]). However, at the time of this writing, no ML-based proposal has succeeded to replace long-established TE solutions; indeed, the best performing TE optimizers to date are based on traditional optimization algorithms, such as Constraint Programming [10], Local Search [9], or Linear Programming [11], [19].

In this paper, we present MAGNNETO, a novel ML framework for distributed TE optimization leveraging Graph Neural Networks (GNN) [20] and Multi-Agent Reinforcement Learning (MARL) [21] at its core[1]. 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.

More in detail, MAGNNETO presents the following contributions:

Top performance with very low execution times: We compare MAGNNETO against a curated set of well-established TE solutions: SRLS [9], DEFO [10] and TabuIGPWO [11]. 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 [22]. Our results show that MAGNNETO 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

G. Bernárdez, J. Suárez-Varela, A. López, P. Barlet-Ros and A. Cabellos-Aparicio are with Barcelona Neural Networking Center, Universitat Politècnica de Catalunya, Barcelona, Spain. Contact: guillermo.bernardez@upc.edu

X. Shi, S. Xiao and X. Cheng are with the Network Technology Lab., Huawei Technologies Co., Ltd., Beijing, China.

[1]MAGNNETO stands for Multi-Agent Graph Neural Network Optimization. The code of this framework and all the data needed to reproduce our experiments are publicly available at: https://github.com/BNN-UPC/Papers/wiki/MAGNNETO-TE.

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* [23]. 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 [23]. 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 [24] 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 [9], [10]. By leveraging Segment Routing [25] 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 paper we show that comparable performance is achievable by using only standard destination-based OSPF routing. Indeed, MAGNNETO is fully compliant with current OSPF-based networks, and does not require the use of any overlay technology.

MAGNNETO is partially based on an earlier version presented at [26]. In that work, we raised an open question: *Is ML ready for Traffic Engineering optimization?* 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 ML framework for TE optimization, and made an exploratory evaluation on this. This paper actually deeps dive into this question by formulating an enhanced ML framework –MAGNNETO– and performing a much more comprehensive evaluation. We summarize below the main novelties of this work with respect to [26]:

- 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 solution [26] operated over a classical MDP, where agents must take actions sequentially in a synchronized manner.
- MAGNNETO 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.
- We present in this paper 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 evaluation of [26] only considered 3 different topologies of limited size (up to 24 nodes), and the results were compared against a single TE solver.

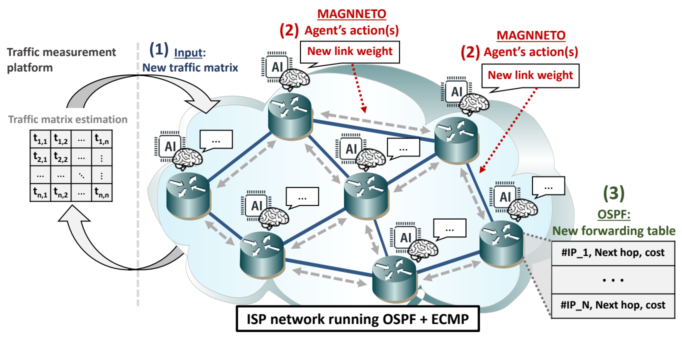

Traffic measurement platform
(1) Input: New traffic matrix
Traffic matrix estimation
$t_{1,1}$ $t_{1,2}$ … $t_{1,n}$
$t_{2,1}$ $t_{2,2}$ … ⋮
… … ⋱ ⋮
$t_{n,1}$ $t_{n,2}$ … $t_{n,n}$
MAGNNETO (2) Agent's action(s)
New link weight
MAGNNETO (2) Agent's action(s)
New link weight
AI
…
(3) OSPF: New forwarding table
#IP_1, Next hop, cost
. . .
#IP_N, Next hop, cost
ISP network running OSPF + ECMP

Figure 1: Intradomain traffic engineering optimization with MAGNNETO.

The remainder of this paper is as follows. Section II describes the TE scenario where we deploy the proposed ML-based system. Section III formalizes MAGNNETO, as a general framework for networked environments. Afterwards, Section IV describes how we adapt this framework to perform intradomain TE optimization. In Section V, we make an extensive evaluation of the proposed framework against state-of-the-art TE proposals. Section VI summarizes the main existing works related to this paper, and lastly Section VII concludes the paper.

II. Network Scenario

This section describes the intradomain TE scenario where MAGNNETO operates. In this paper, we consider the intradomain TE problem, where network traffic is measured and routed to minimize network congestion. Typically, IP networks run link-state Interior Gateway Protocols (IGP), such as Open Shortest Path First (OSPF) [12], that choose 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 [27]. Network operators commonly use commercial tools [28], [29] to fine-tune link weights. However, other mechanisms propose to add extra routing entries [30] or end-to-end tunnels (e.g., RSVP-TE [31]) 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 [32].

Figure 1 illustrates the general operational workflow of MAGNNETO:

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. 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 paper 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 [11]), where the TM is first estimated and then optimized. Some proposals trigger the optimization process when a relevant change is detected in the TM [33], while others use prediction techniques to optimize it in advance [34]. 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. 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 [7], [8], [10]. This problem is known to be NP-hard, and even good settings of the weights can deviate significantly from the optimal configuration [8], [32]. Our MARL optimization system is built using a distributed Graph Neural Network (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 III-V.

3) OSPF convergence: Lastly, the standard OSPF convergence process is executed taking into account the new per-link weights computed by MAGNNETO. 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 Advertisements (LSAs) [12]. Once the routers have an identical view of the network, they compute locally their new forwarding tables (Fig. 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 [35].

III. MAGNNETO

This section provides a detailed description on how MAGNNETO operates. To do so we first briefly introduce the main ML methodologies it implements. Note that MAGNNETO is conceived as a general framework to optimize networked environments in a distributed fashion; details on how it is particularly adapted to face intradomain TE are then provided in Section IV.

A. Related ML-based Technologies

MAGNNETO incorporates two well-known ML-based mechanisms: Multi-Agent Reinforcement Learning and Graph Neural Networks. Let us provide some background on these technologies:

1) Reinforcement Learning (RL): According to the regular setting of RL [36], 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 an episodic, time-homogeneous Markov Decision Process (MDP) $(\mathcal{S}, \mathcal{A}, r, P, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are respectively the state and action spaces; P is the transition kernel, $s_{t+1} \sim P(\cdot|s_t, a_t)$; r_t represents the immediate reward given by the environment after taking action a_t from state s_t; and $\gamma \in (0, 1]$ is the discount factor used to compute the return G_t, defined as the –discounted– cumulative reward from a certain time-step t to the end of the episode T: $G_t = \sum_{t=0}^{T} \gamma^t r_t$. The behavior 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, and the goal of an RL agent is to find the optimal policy in the sense that, given any considered state $s \in \mathcal{S}$, it always selects an action that maximizes the expected return $\hat{G}_t$. There are two main model-free approaches to this end [37]:

- Action-value methods, typically referred to as Q-learning; the policy π is indirectly defined from the learned estimates of the action value function $Q^{\pi}(s, a) = \mathbb{E}_{\pi} [G_t | s_0 = s, a_0 = a]$.
- 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).

2) Multi-Agent Reinforcement Learning (MARL): 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 MDP (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 reward r_t), and it evolves to a next global state $s_{t+1} \in \mathcal{S}$ –i.e., each agent v transitions to 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 [38].

3) Graph Neural Networks (GNN): These models are a recent family of neural networks specifically conceived to operate over graph-structured data [20], [23]. Among the numerous GNN variants developed to date [39], we focus on Message Passing Neural Networks (MPNN) [40], which is a well-known type of GNN whose operation is based on an iterative message-passing algorithm that propagates information between elements in a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. Focusing on the set of nodes, the process is as follows: first, each node $v \in \mathcal{N}$ initializes its hidden state h_v^0 using some initial features already included in the input graph. At every message-passing step k, each node v receives via messages the current hidden state of all the nodes in its neighborhood $\mathcal{B}(v) = \{u \in \mathcal{N} \,|\, \exists e \in \mathcal{E}, e = (u, v) \vee e = (v, u)\}$, and processes them individually by applying a message function *m(·)* together with its own internal state h_v^k. Then, the processed messages are combined by an aggregation function *a(·)*:

$$M_v^k = a(\{m(h_v^k, h_i^k)\}_{i \in \mathcal{B}(v)}) \tag{1}$$

Finally, an update function *u(·)* is applied to each node v; taking as input the aggregated messages M_v^k and its current hidden state h_v^k, it outputs a new hidden state for the next step $(k+1)$:

$$h_v^{k+1} = u(h_v^k, M_v^k). \tag{2}$$

After a certain number of message passing steps K, a readout function *r(·)* takes as input the final node states h_v^K to produce the final output of the GNN model. This readout function can predict either features of individual elements (e.g., a node's class) or global properties of the graph. Note that a MPNN model generates *a single set of message, aggregation, update, and readout functions that are replicated at each selected graph element.*

B. Execution Framework

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 RL-based function that is aimed to learn –i.e., the global state-action value function Q_θ for the former, or the global policy π_θ for the latter.

A main contribution 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 MDP of single-agent RL [37]. On the other hand, it requires the more challenging Dec-POMDP formalization of standard MARL [21] when letting several agents act simultaneously. 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 MPNN (see Sec. III-A3), 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 1 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 III-A3.

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,

Algorithm 1: 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 $X_{\mathcal{G}}^0$, episode length $T$,
       number of message passing steps $K$
1  Agents initialize their states $s_v^0$ based on $X_{\mathcal{G}}^0$
2  for $t \leftarrow 0$ to $T$ do
3      Agents initialize their hidden states
         $h_v^0 \leftarrow (s_v^t, 0, \ldots, 0)$
4      for $k \leftarrow 0$ to $K$ do
5          Agents share their current hidden state $h_v^k$ to
             neighboring agents $\mathcal{B}(v)$
6          Agents process the received messages
             $M_v^k \leftarrow a_{\theta_a}(\{m_{\theta_m}(h_v^k, h_\mu^k)\}_{\mu \in \mathcal{B}(v)})$
7          Agents update their hidden state
             $h_v^{k+1} \leftarrow u(h_v^k, M_v^k)$
8      end for
9      Agents partially evaluate the RL function $f_\theta$ over
         their own actions $\{f_\theta(s_t, a)\}_{a \in \mathcal{A}_v} \leftarrow r_{\theta_r}(h_v^K)$
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
         $X_{\mathcal{G}}^{t+1}$
13     Agents update their states $s_v^{t+1}$ based on $X_{\mathcal{G}}^{t+1}$
14 end for
Output: New graph configuration $X_{\mathcal{G}}^*$ that optimizes
        some pre-defined objective or metric
```

or directly the q-value associated to each action when considering Q-learning algorithms. The second and last steps involve 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 $s_t = \prod_{v \in \mathcal{V}} s_v^t$ 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 does 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.

IV. MAGNNETO for Traffic Engineering

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

A. General Setting

A straightforward approach to map the graph $\mathcal{G}$ of the described MAGNNETO framework 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 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) [41], which offers a favorable balance between reliability, sample complexity, and simplicity. Consequently, in this case the global function f_θ of the framework (see Sec. III-B) is the global policy π_θ of the actor. Regarding the critic's design, more information can be found in Section IV-C.

B. Adapting MAGNNETO to TE

Having clear the general configuration of our MAGNNETO 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:

1) 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 [11]). 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. II).

2) State Space and Message Passing: At each time-step t of an episode, each link-based agent $v_e \in \mathcal{V}$, feeds its MPNN module with its input features x_e^t to generate its respective initial hidden state h_e^0 (Figure 2.a). In particular, agents consider

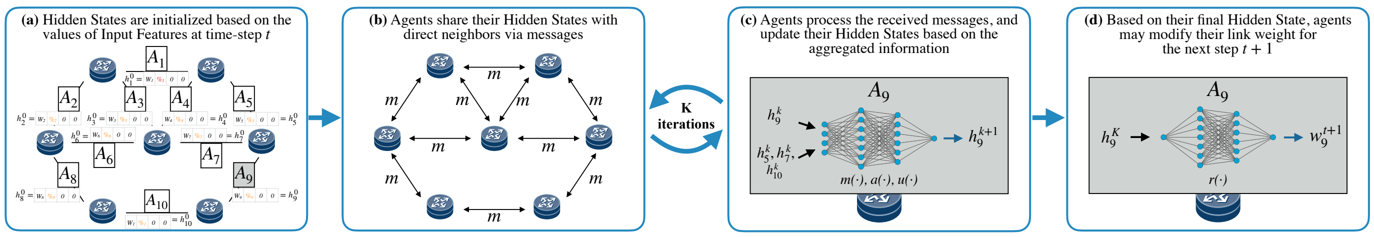

(a) Hidden States are initialized based on the values of Input Features at time-step t
A_1
A_2
A_3
A_4
A_5
A_6
A_7
A_8
A_9
A_{10}
(b) Agents share their Hidden States with direct neighbors via messages
m
K iterations
(c) Agents process the received messages, and update their Hidden States based on the aggregated information
h_9^k
h_5^k, h_7^k, h_{10}^k
h_9^{k+1}
$m(\cdot), a(\cdot), u(\cdot)$
(d) Based on their final Hidden State, agents may modify their link weight for the next step $t+1$
h_9^K
w_9^{t+1}
$r(\cdot)$

Figure 2: Description of the message passing and action selection process of MAGNNETO 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.

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 2.b and 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.

3) 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 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 III-B, 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 $t+1$.

4) 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.

C. 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 III-A1, 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 a centralized design would have no impact on the distributed nature of MAGNNETO.

In fact, following the standard approach of MARL systems [38], 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 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 [41]. The same process of generating episodes and updating the model is repeated for a fixed number of iterations to guarantee convergence.

V. Evaluation

In this section we extensively evaluate MAGNNETO in an intradomain TE scenario: we benchmark it against a curated set of advanced TE optimizers in more than 75 different real-world topologies, using realistic traffic loads. As shown in our experimental results, MAGNNETO achieves similar performance to state-of-the-art TE optimizers with a significantly lower execution time. 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.

A. Baselines

In this section we describe the set of baselines we use to benchmark MAGNNETO 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 Equal-Cost Multi-Path (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 [22]:

- TabuIGPWO (IGP Weight Optimizer, based on [11]): This algorithm runs a Local Search 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 (i.e., OSPF link weight configuration).
- DEFO (Declarative and Expressive Forwarding Optimizer) [10]: It uses Constraint Programming and Segment Routing (SR) [25] 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 (Segment Routing and Local Search) [9]: By leveraging Local Search 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 and TabuIGPWO operate directly over standard OSPF-based networks with destination-based routing.

B. 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 [42], and 79 networks from the Internet Topology Zoo dataset [24]. In this section we provide more low-level technical details of MAGNNETO'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. V-C). 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.

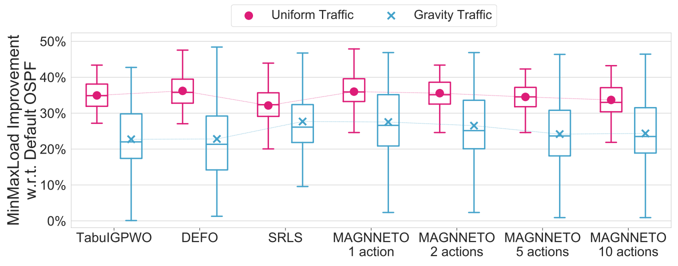


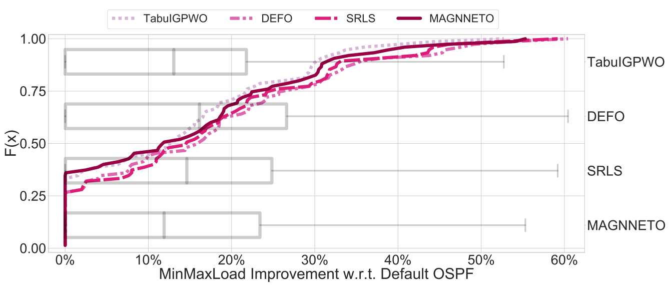
Figure 3: Evaluation of MAGNNETO for different number of simultaneous actions $n \in \{1, 2, 5, 10\}$, each of them considering an episode length of $T = 150/n$. The training only considers samples of NSFNet and GEANT2 topologies, and the evaluation is performed over 100 unseen TMs on the GBN topology. 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.

Taking [43] 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.

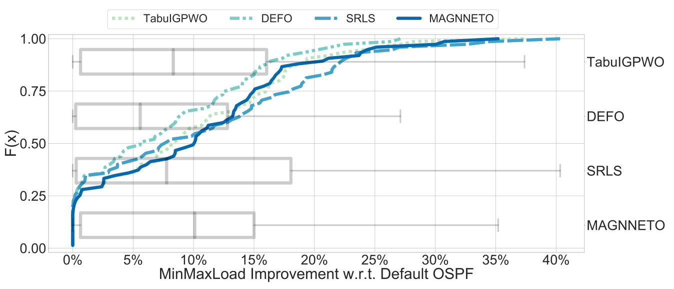
For each experiment, we generate two sets of simulated traffic matrices: uniform distribution across source-destination traffic demands, and traffic distributions following a gravity model [44] –which produces realistic Internet traffic patterns. The training process of MAGNNETO highly depends on the 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).

C. Multiple Actions and Episode Length

As previously mentioned in Section III, 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.



(a) TopologyZoo Uniform



(b) TopologyZoo Gravity

Figure 4: Evaluation of MAGNNETO'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 [24], and boxplots are computed based on these mean improvement values as well. Both the uniform (a) and gravity (b) MAGNNETO models evaluated were trained exclusively on samples from the NSFNet and GEANT2 topologies [42].

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 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 our preliminary work [26]. However, whereas [26] was subject to n=1 by design, MAGNNETO 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 3 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 V-F we will analyze in detail the execution cost of MAGNNETO.

D. Generalization over Unseen Topologies

In Section I 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 under an intensive evaluation in this regard.

In our experiments, MAGNNETO only observes NSFNet (14 nodes, 42 links) and GEANT2 (24 nodes, 74 links) samples during training [42], whereas the evaluation is performed over a subset of 75 networks from the Topology Zoo dataset [24] 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.

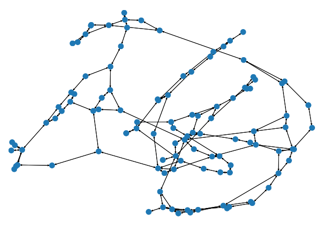
Overall, this experimental setup comprises 7, 500 evaluation runs for each traffic profile, which we summarize in Figures 4a and 4b, respectively for uniform and gravity-based loads. In particular, note that we first compute the mean *MinMaxLoad* improvement of MAGNNETO –and the baselines– over the 100 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.

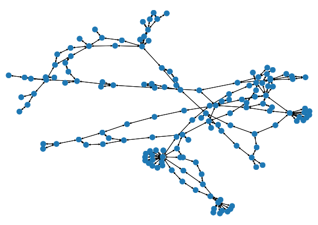
In both traffic scenarios MAGNNETO achieves comparable performance to the corresponding best performing benchmark –DEFO when considering uniform traffic and SRLS for gravity. In fact, MAGNNETO outperforms TabuIGPWO, improves DEFO with gravity-based traffic, and lies within a 2% average improvement difference 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.

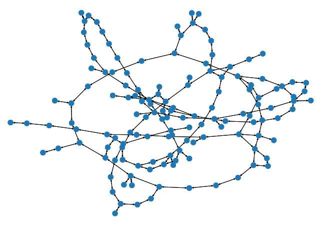
E. Traffic Changes in Large Topologies

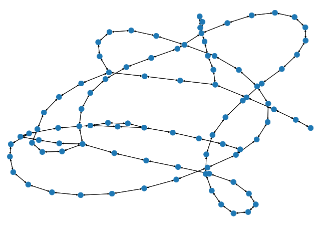
After evaluating the generalization capabilities of MAGNNETO, we aim to test the performance of our method over traffic changes in large networks, where the combinatorial of the optimization process might dramatically increase. Having considered networks up to 30 nodes and 90 links so far, for this set of experiments we arbitrarily select four large real-world topologies from Topology Zoo [24]: Interoute (110 nodes, 294 links), Colt (153 nodes, 354 links), DialtelecomCz (138 links, 302 links) and VtlWavenet2011 (92 nodes, 192 links). Figures 5.I-IV depict these topologies.

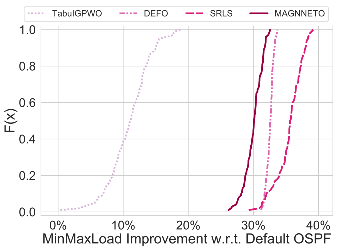
In these experiments, for each traffic profile (uniform or gravity) we train a MAGNNETO model on each network. Then, we evaluate models on the same topology where they


I. Interoute

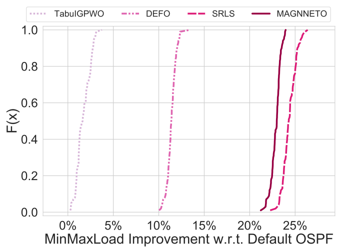

II. Colt


III. DialtelecomCz

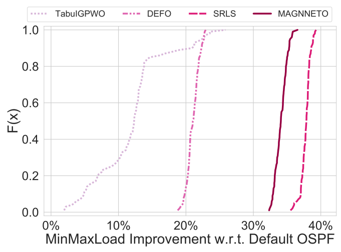

IV. VtlWavenet2011



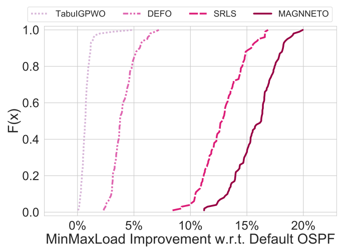
(a) Interoute Uniform



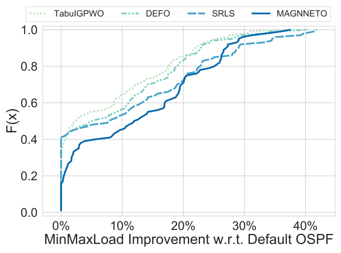
(b) Colt Uniform



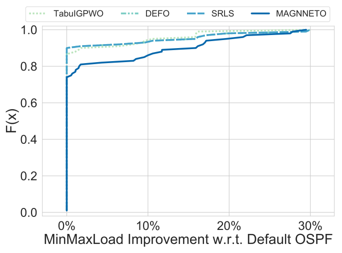
(c) DialtelecomCz Uniform



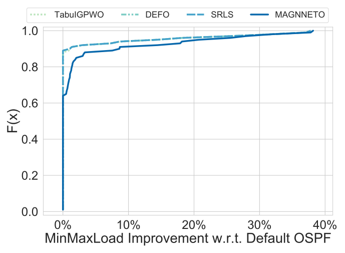
(d) VtlWavenet2011 Uniform



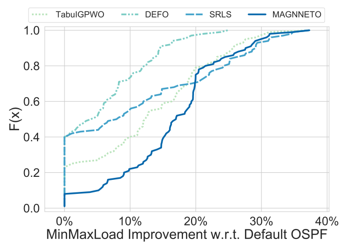
(e) Interoute Gravity



(f) Colt Gravity



(g) DialtelecomCz Gravity



(h) VtlWavenet2011 Gravity

Figure 5: Evaluation of MAGNNETO on traffic changes in four large real-world topologies (I-IV) from the Topology Zoo dataset [24], both for uniform ((a)-(d)) and gravity-based ((e)-(h)) traffic loads. A MAGNNETO 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.

were trained, over 100 different TMs not previously seen in training. Figures 5.a-d and 5.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 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 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 attained equivalent –or even better– performance than the advanced TE optimizers benchmarked. These results evince its potential to successfully operate in large computer networks.

F. Execution Cost

Lastly, in this section we evaluate the execution cost of MAGNNETO. 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 I gathers these results for several variable-sized networks used in the previous evaluations.

Taking into account the recommendations of REPETITA [9], as well as analyzing the results provided in the original works [9]–[11], 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 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 V-C these times could be further reduced by allowing multiple simultaneous actions. For instance, by considering up to 10 simultaneous actions, MAGNNETO 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's distributed execution naturally parallelizes the global optimization process across all network devices (i.e.,

	NSFNet	GBN	GEANT2	VtlWavenet2011	Interoute	DialtelecomCz	Colt
(#nodes, #links)	(14,42)	(17,52)	(24,74)	(92,192)	(110,294)	(138,302)	(153,354)
MAGNNETO Link Overhead* (MB/s)	1.20	1.32	1.20	0.83	1.28	0.91	1.01
Execution Time (s)							
TabuIGPWO [11]	600	600	600	600	600	600	600
DEFO [10]	180	180	180	180	180	180	180
SRLS [9]	60	60	60	60	60	60	60
MAGNNETO [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 I: Cost of MAGNNETO: Average link overhead and execution time –in terms of the maximum number of simultaneous actions allowed– for variable-sized network topologies.

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 I shows that the link overhead produced by MAGNNETO (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 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.

VI. Related Work

Recently, numerous solutions based on Deep Reinforcement Learning (DRL) have been proposed to solve complex networking problems, especially in the context of routing optimization and TE [15], [17], [45]. 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 [16], the authors design a DRL-based architecture that obtains better results than Shortest Path and Load Balancing routing. Regarding MARL-based solutions [46], [47], most of them suffer from the same lack of topology generalization. An exception to that is the work of [18], 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 [20], [48], and in particular Message Passing Neural Networks (MPNN) [40], 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 [42] 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 [49] 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 [50], 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 [51], [52] are theoretical papers from the ML community, and none of them apply to the field of networking.

VII. Conclusions

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 [10], SRLS [9]), 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 [15]–[18]. However, at the time of this writing no ML-based solution had shown to outperform state-of-the-art TE optimizers.

In this paper we have presented MAGNNETO, a novel ML-based framework for intradomain TE optimization. Our system implements a novel distributed architecture based on Multi-Agent Reinforcement Learning and Graph Neural Networks. In our evaluation, we have compared MAGNNETO 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 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 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 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).

Acknowledgment

This publication is part of the Spanish I+D+i project TRAINER-A (ref. PID2020-118011GB-C21), funded by MCIN/ AEI/10.13039/501100011033. This work is also partially funded by the Catalan Institution for Research and Advanced Studies (ICREA), the Secretariat for Universities and Research of the Ministry of Business and Knowledge of the Government of Catalonia, and the European Social Fund.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, "Requirements for traffic engineering over MPLS," RFC 2702, 1999.

[6] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao, "Overview and principles of internet traffic engineering," RFC 3272, 2002.

[7] 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.

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

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

[10] 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.

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

[12] J. Moy, "Ospf version 2," STD 54, RFC Editor, April 1998.

[13] 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.

[14] 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.

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

[16] 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)*, pp. 1871–1879, 2018.

[17] 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.

[18] 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)*, pp. 1–11, 2020.

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

[20] 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.

[21] J. N. Foerster, *Deep multi-agent reinforcement learning*. PhD thesis, University of Oxford, 2018.

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

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] Cisco, "Planning and designing networks with the cisco mate portfolio." white paper, 2013.

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

[30] 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.

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

[32] 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.

[33] 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*, pp. 1–12, 2011.

[34] M. Luo, "Dsox: Tech report," tech. rep., Huawei Shannon Lab, 2013.
[35] A. Basu and J. Riecke, "Stability issues in ospf routing," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 225–236, 2001.
[36] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-dynamic programming*, vol. 5. Athena Scientific Belmont, MA, 1996.
[37] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[38] 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, pp. 289–353, May 2008.
[39] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
[40] 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*, pp. 1263–1272, 2017.
[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[42] 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)*, pp. 140–151, 2019.
[43] 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.
[44] M. Roughan, "Simplifying the synthesis of internet traffic matrices," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 93–96, 2005.
[45] 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)*, pp. 1–6, 2019.
[46] 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)*, pp. 932–937, IEEE, 2020.
[47] 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.
[48] 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, pp. 729–734, IEEE, 2005.
[49] 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.
[50] 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)*, pp. 40–45, 2018.
[51] J. Jiang, C. Dun, T. Huang, and Z. Lu, "Graph convolutional reinforcement learning," in *International Conference on Learning Representations*, 2020.
[52] J. Su, S. Adams, and P. A. Beling, "Counterfactual multi-agent reinforcement learning with graph convolution communication," *arXiv preprint arXiv:2004.00470*, 2020.