Fast Traffic Engineering by Gradient Descent with Learned Differentiable Routing

Krzysztof Rusek[†], Paul Almasan[*], José Suárez-Varela[*], Piotr Chołda[†], Pere Barlet-Ros[*],
Albert Cabellos-Aparicio[*]

[†]AGH University of Science and Technology, Institute of Telecommunications
[*]Barcelona Neural Networking Center, Universitat Politècnica de Catalunya, Spain

Abstract—Emerging applications such as the metaverse, telesurgery or cloud computing require increasingly complex operational demands on networks (e.g., ultra-reliable low latency). Likewise, the ever-faster traffic dynamics will demand network control mechanisms that can operate at short timescales (e.g., sub-minute). In this context, Traffic Engineering (TE) is a key component to efficiently control network traffic according to some performance goals (e.g., minimize network congestion).

This paper presents *Routing By Backprop* (RBB), a novel TE method based on Graph Neural Networks (GNN) and differentiable programming. Thanks to its internal GNN model, RBB builds an end-to-end differentiable function of the target TE problem (*MinMaxLoad*). This enables fast TE optimization via gradient descent. In our evaluation, we show the potential of RBB to optimize OSPF-based routing ($\approx$25% of improvement with respect to default OSPF configurations). Moreover, we test the potential of RBB as an initializer of computationally-intensive TE solvers. The experimental results show promising prospects for accelerating this type of solvers and achieving efficient online TE optimization.

Index Terms—Traffic Engineering, Routing Optimization, Graph-based Deep Learning, Gradient Descent

I. INTRODUCTION

Communication networks are witnessing a fast-paced transformation driven by the continuous emergence of new services and applications. Novel trends such as the metaverse (AR/VR), cloud computing, vehicular networks, or telesurgery require increasingly complex operational demands. This puts great pressure on network operators, which need to count on more advanced network mechanisms that can quickly react to the fast network dynamics (e.g. traffic changes, link failures), as well as guarantee hard service-level requirements (e.g. ultra-reliable low latency) [1]. For example, recent studies in Wide Area Networks show that traffic can change dramatically at the scale of a few minutes ($\approx$1-5 mins) [2], [3]. This motivates the need for fast network optimizers that can operate at sub-minute timescales.

In this paper, we revisit Traffic Engineering (TE), a fundamental networking task whose goal is to optimize the network performance and resource utilization. TE is a broad topic that comprises a wide spectrum of problems with different optimization goals, and using various traffic control mechanisms (e.g., routing, queue scheduling) [4]–[7]. We focus on the classical problem of minimizing the maximum link load in the network (*MinMaxLoad*) by optimizing the configuration of OSPF. This is an NP-hard optimization problem [5].

Given the relevance of this problem, a large body of solutions can be found in the literature [6]–[8]. Existing TE solutions can be classified in two main blocks. On the one hand, some solutions rely on general-purpose mathematical programming methods (e.g., using Local Search, Constraint Programming [6], [8]). These solutions can typically produce near-optimal results with performance and time guarantees. However, they are computationally intensive, which makes them unsuitable for online optimization in modern real-world networks. On the other hand, networking experts have developed over the years efficient handcrafted heuristics for TE [9]. These methods can typically operate faster and scale effectively to large real-world networks. As a counterpart, heuristics typically lead to lower performance compared to the previous TE solvers based on general-purpose mathematical programming methods [6], [8].

Our main goal is to design a TE solution that can operate as fast as traditional heuristics, while achieving comparable performance to computationally-intensive TE solvers. In this paper, we present *Routing by Backprop* (RBB), a novel method for online TE leveraging Graph Neural Networks (GNN) [10] and differentiable programming at its core. The code used in our experiments is publicly available, including a short demo[1]. Given a network topology and an estimated traffic matrix, RBB outputs a set of link weights that optimize the configuration of the OSPF protocol (widely deployed nowadays in networks [9], [11]). To produce these weights, RBB internally implements a GNN-based model that approximates the target TE problem as an end-to-end differentiable function. The resulting GNN-based model introduces a continuous relaxation of the target optimization function. Thus, it is suitable for fast TE via differentiable programming, i.e., by applying a common backpropagation method and optimizing through gradient descent.

---

[1]https://github.com/krzysztofrusek/net2vec

In our evaluation, we first apply RBB to the *MinMaxLoad* TE problem in four different real-world networks unseen during the training phase. Our results show that — by optimizing the OSPF link weights — the proposed solution reduces the maximum link load by $\approx$25% on average, compared to the default OSPF configurations widely deployed by network operators nowadays [11]. At the same time, RBB operates at sub-second timescales (median $<$322 ms in networks up to 37 nodes), thus showing promising results for fast TE.

Likewise, we explore the potential of RBB to be utilized as an initializer of computationally-intensive TE solvers. In our experiments, we take as a reference IGP-WO, a popular TE solver based on Local Search [8]. We limit its execution time to only 1 second. As a reference, in [8] the authors report 5-minute executions to optimize networks of similar size with this TE solver. Our experimental results show that, by initializing IGP-WO with the output produced by RBB, we can quickly reduce $\approx$30% on average the maximum link load with respect to default OSPF settings. This further reveals the potential of the proposed solution to accelerate advanced TE solvers, and thus meet the hard time constraints of online TE.

II. PROPOSED SOLUTION: RBB

This section presents the mathematical framework behind Routing by Backprop (RBB). For the mathematical definitions, we treat the network topology as a graph. Then, we use GNNs and soft routing to produce optimized link weights that minimize the maximum link utilization. Lastly, these weights are used by OSPF to update the routing tables on devices, by computing the weighted shortest path (i.e., Dijkstra's algorithm).

Dijkstra's algorithm is an exact deterministic procedure that maps a weighted graph (e.g., weights on the edges) to a set of shortest paths (i.e., one path for each pair of nodes). This can be seen as a function that maps a vector with weights and a network topology to a set of shortest paths. In our work, we rely on the universal approximation property of neural networks to approximate this function. Specifically, we use GNNs as the universal approximation function, enabling optimization on any continuous function of the shortest path using the gradient with respect to the weights input vector.

A. Notation

We represent the network as a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{r_i\}_{i=1:n_v}$ is a set of routers (nodes) and $\mathcal{E} = \{(e_k, r_k, s_k)\}_{k=1:n_e}$ is a set of links (edges) connecting receiver and sender nodes indexed by r_k, s_k. Both, routers and links, are described by their properties (features) respectively denoted as r_i (node features in ML terminology) and e_i (edge features).

Routing in the network can be considered as a mapping from a pair of routers to the path connecting them. Let u and v be the indices of two routers, then the path can be represented as a binary vector $p_{u,v} \in \{0,1\}^{n_e}$ with 1 on the k-th position if the path contains k-th link. In this paper we use shortest path routing so these are parameterized by

positive link weights (costs) $w = [w_k]_{k=1:n_e}$ such that the path minimizes $p_{u,v}w^T$ [5]. In this way, the weights indirectly control link load, since each path $p_{u,v}$, has an associated traffic demand volume $d_{u,v}$ routed through it.

Link utilization vector ρ can be expressed with a convenient matrix operation. Let $u \in \mathbb{N}^{n_p}$ and $v \in \mathbb{N}^{n_p}$ be the vectors representing the indices of all possible router pairs ($n_p = n_v(n_v - 1)$). Similarly $P \in \{0,1\}^{n_p \times n_e}$ is a matrix that stacks paths for those pairs $P = [p_{u_i,v_i}^T]_{i=1:n_p}^T$. P allows us to compute the utilization of every link as $\rho = d \cdot P \odot \frac{1}{c}$, where $d = [d_{u_i,v_i}]_{i=1:n_p}$ and c_i is the capacity of the i-th link. Under common assumption of negligible losess [7], the *MinMaxLoad* problem can be described as:

$$w_{\text{opt}} = \arg\min_{w} \max(\rho(w)). \tag{1}$$

RBB proposes a method for relaxing (1) so that a gradient optimization method can be applied.

B. Soft Routing

Differentiable programming was previously discussed in [12] for TE. In that paper, the authors propose to use a deep learning model that makes directly end-to-end TE decisions based on a set of candidate shortest paths. Instead, we consider a soft routing approach based on link weights that makes the gradient more meaningful for gradient-based optimization methods. For this purpose, we use a GNN model that introduces a continuous relaxation on the target step-wise optimization function (where the true gradient is zero in most regions). Moreover, this link-based routing representation is directly compatible with OSPF, which is nowadays the dominant routing protocol in networks [9], [11].

We leverage the property that if we set $e_k = [w_k]$ and $r_i = [I(u = i), I(v = i)]$ we end up with the path being a function of graph data ($e_k \in \mathbb{R}^+, r_i \in \{0,1\}^2$). Such a function can be approximated by a GNN trained on the routing obtained by the Dijkstra's algorithm (i.e., shortest path). A GNN is differentiable, however the final discrete path $p_{u,v}$ is not, so we approximate the path by the probabilistic output from the GNN, yet being differentiable:

$$\hat{p}_{u,v} = \mathsf{P}(p_{u,v} = 1) = f(\mathcal{G}(w, u, v)). \tag{2}$$

Note that (2) is defined only for a single path. However, leveraging recent developments in Machine Learning (ML) libraries, we can easily vectorize (2) into the soft routing matrix $\hat{P}(w) = \text{vmap} f(\mathcal{G}(w, u, v))$ and compute all paths in a single evaluation of batched GNN — e.g., using the vmap transform from JAX [13]. From this approximation we get the link utilization as:

$$\hat{\rho}(w) = d \cdot \hat{P}(w) \odot \frac{1}{c}. \tag{3}$$

At this point we could optimize (1), as the $\max$ function is differentiable. Nevertheless, we observe that using softmax can be beneficial to achieve more relaxation on the target optimization goal. In particular, we use softmax with temperature

τ that controls its steepness (relaxation) to approximate the max function:

$$\widetilde{\max}(\boldsymbol{x}) \equiv \frac{\sum_i x_i e^{\frac{x_i}{\tau}}}{\sum_j e^{\frac{x_j}{\tau}}}. \qquad (4)$$

Combining (3) with (4) in the gradient descent algorithm, we propose the following weight update rule that minimizes the maximum utilization:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \alpha_t \nabla \widetilde{\max}\left(\hat{\boldsymbol{\rho}}(\boldsymbol{w}_t)\right), \qquad (5)$$

where $\alpha_t > 0$ is the learning rate. Here we rely on automatic differentiation to compute $\nabla \widetilde{\max}\left(\hat{\boldsymbol{\rho}}(\boldsymbol{w}_t)\right)$ [14]. Note that we are computing gradients with respect to the model inputs instead of the internal GNN weights, so we cannot reuse the gradients used for training the GNN model.

C. Graph Neural Network

In the proposed method, the only trainable part is the relaxed shortest path calculation from link weights in (2), which is implemented by a GNN. Further calculations are based on prior knowledge from existing analytical network models [9]. Below we elaborate on the design of our GNN model to approximate (2). We refer the reader to [10] for generic background on GNNs.

We adopt a highly flexible architecture named encode-process-decode [10]. Such an architecture consists of an encoder model that independently encodes node and edge attributes (e_i and r_i). A core model, which performs T rounds of message-passing steps and a decoder that independently decodes the resulting node and edge attributes.

The encoder and decoder in our proposal are implemented with 2-layer dense neural networks followed by a normalization layer. The processor consists of T graph network blocks [10]. Each block updates the edge representation by applying a dense neural network f^e (similar to the encoder) to the concatenation of the edge, sender, and receiver node attributes: $e_k^{t+1} = f^e([e_k^t, r_{r_k}^t, r_{s_k}^t])$. The attributes of all edges adjacent to a given node are aggregated by a function, which in our case is a summation: $\bar{e}_i^t = \sum_{k: r_k = i} e_k^t$. Given the information from the neighboring edges, the node representation is updated by another neural network f^v: $r_i^{t+1} = f^v([\bar{e}_i^t, r_i^t])$.

The final output of the GNN is the result of the decoder network being independently applied to all node and edge attributes. In our case, we only generate edge-level outputs, which represent the probability that a given link belongs to a shortest path. The resulting GNN was trained by backpropagation using cross entropy as a loss function between predictions and the true values from the Dijkstra's algorithm. We treat it as a classification problem. Particularly, we use the loss computed at each message-passing step and backpropagate the sum. In this way, each graph network block learns how to iteratively improve upon the previous one, which introduces a bias that may help with faster convergence [10].

Lastly, we combine this GNN model with the soft routing approach of Section II-B. Particularly, the GNN model acts

TABLE I
PERCENTAGE OF CASES WHERE RBB OUTPERFORMS DEFAULT OSPF

Network	n_v (nodes)	n_e (links)	$\overline{\max \boldsymbol{\rho}_0 > \max \boldsymbol{\rho}_3}$
cost266	37	57	100%
geant	22	36	99.4%
janos-us	26	84	100%
nobel-germany	17	26	99.9%

as a differentiable surrogate method for $\hat{\boldsymbol{P}}(\boldsymbol{w})$ in Equation (3). This makes it possible to build an end-to-end differentiable function that models the target TE problem (i.e., MinMaxLoad).

III. EVALUATION

In this section, we evaluate RBB under a wide variety of network topologies and traffic matrices. We also test RBB's potential for accelerating high-performance TE solvers based on general-purpose mathematical programming.

In the experiments we consider that all links have a capacity of 1 Mbps and traffic matrices are generated using a uniform distribution. Specifically, for each node pair, the traffic generated is uniformly sampled from the interval (0, *n-1*), with *n* being the number of nodes in the graph. The resulting traffic is then injected into the network following the shortest path routing (i.e., using the link-level weights). We scale the traffic matrix values by a topology-specific number in such a way that the link load distribution has a soft tail above 1 Mbps (i.e., there are a few overloaded links).

A. Training of the GNN model

To train the GNN model, we first generate a dataset with random topology graphs using the Barabási-Albert (BA) model [15]. The number of nodes was randomly sampled between the range $[10, 20]$. In all topologies we sampled the link weights from a scaled beta distribution in the range $(0, 2)$ with mode 1. For each training sample, we selected two random points (src, dst) and computed the weighted shortest path between them using Dijkstra's algorithm.

All components of the encode-process-decode GNN architecture were implemented with a 2-layer MLP with 128 units, each followed by a normalization layer. The output of the decode function uses a sigmoid activation to produce link probabilities in the range $(0, 1)$. The model was trained using Adam optimizer with learning rate of 0.001 and with batches of 32 graphs. For the validation, we considered 2,000 samples of 4 real-world network topologies extracted from SNDLib [16] and described in Table I.

We observed loss convergence after 8,000 training steps. In our setup, the whole training process took $\approx$51 min. At this point, the model achieved a link prediction accuracy of 0.9893 ± 0.0005 on the validation samples. Once the model was trained, we used it as a differentiable surrogate for shortest path routing to implement the optimization method described in Section II.
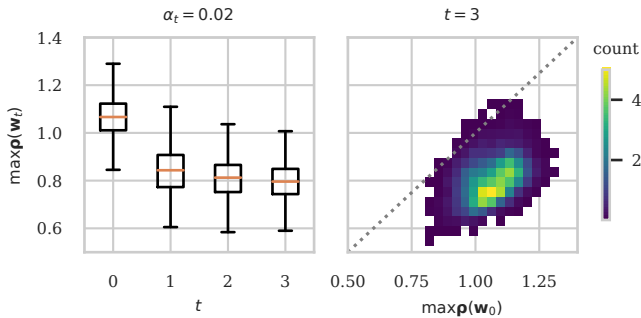
Fig. 1. Comparison with *Default OSPF* in 12,000 samples from the 4 real-world topologies of Table I. Left: Evolution of the maximum link utilization (y-axis) per optimization step with RBB (x-axis). $t=0$ represents the results of the initial *Default OSPF* configuration. Right: Joint histogram of the maximum link utilization. X-axis represents the results of *Default OSPF*, and the y-axis shows the results after three optimization steps with RBB.

Fig. 2. Comparison between Default OSPF, RBB and IGP-WO in 4 real-world topologies (each with 3,000 samples with different traffic matrices).

B. Comparison with Default OSPF

In this section, we explore the capabilities of RBB to optimize the routing configuration in OSPF. Specifically, RBB produces new link weights according to Equation (5) so that the maximum link utilization is minimized. For each experiment, we considered a topology from Table I where the traffic matrix was sampled from a uniform distribution. We initialized the link weights inversely proportional to the link capacity, which is a default configuration commonly adopted by operators [11]. Hereafter, we refer to this baseline as *Default OSPF*. In the optimizer, we set $\tau = 0.1$ for the softmax function in Equation (4), and $\alpha = 0.02$ in Equation (5).

We evaluated RBB on 12,000 samples from the 4 real-world topologies in Table I (i.e., 3,000 samples from each topology with different traffic matrices). Figure 1 (left) shows the results for the first three optimization steps ($t=0$ corresponds to the initial *Default OSPF* configuration). The results indicate that it takes only one optimization step ($t=1$) to reduce the maximum link utilization below 1 in over 93% of the cases. Likewise, after three optimization steps RBB avoids overloaded links (i.e., utilization<1) in 99.8% of the cases. On average, the maximum link utilization is reduced by $\approx$**25%** with respect to *Default OSPF* after three optimization steps ($t=3$).

In Figure 1 (right) we observe the joint histogram of initial and final maximum link utilizations after three optimization steps. Most of the experiments are far below the diagonal line, indicating that the final configuration produced by RBB considerably outperforms *Default OSPF*. Table I (right column) shows the portion of cases where RBB outperforms *Default OSPF* in each network. It can happen — although rarely ($< 0.6\%$) — that the final configuration is worse than the initial one. Nevertheless, in a practical application the final configurations could be quickly verified before deployment so that in those rare cases *Default OSPF* could be used instead.

C. Execution Time

In the proposed method, the GNN model is evaluated for each source-destination pair to account for the whole routing,
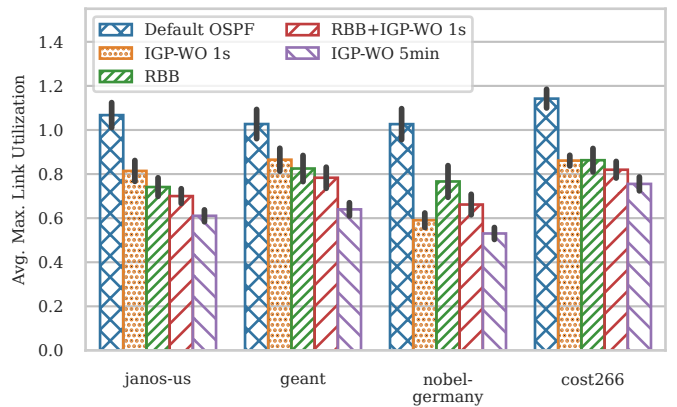
so the numerical complexity is quadratic $O(n_v^2)$. To obtain the highest possible performance we leveraged the JAX library that offers automatic differentiation, vectorization, and a JIT compiler based on XLA [13].

First, the GNN is vectorized to compute all paths at once. Then, the model is transformed to compute the gradient of the softmax of the link utilization with respect to the link weights. Finally, the whole operation is compiled for a GPU (Tesla V100-SXM2-32GB in our experiments). Table II shows the resulting execution times for one optimization step (median of 3,000 repetitions, confidence interval is less than the least significant digit). For larger topologies, it would be possible to distribute the computation of different source-destination pairs across multiple GPUs.

D. Bootstrapping high-performance TE solvers

In this section we test the potential of RBB to act as an initializer to other computationally-intensive TE methods. To this end, we combined RBB with a state-of-the-art link weight optimization algorithm called *IGP-WO* [8], which is based on Local Search. A main advantage of IGP-WO is that it is an anytime optimization method, meaning that the optimization process can be stopped at any moment and the result obtained is valid [7].

Similarly to Section III-B, we executed three optimization steps with the RBB method. The resulting link weights were used to initialize the IGP-WO method (hereafter, referred to as *RBB+IGP-WO*). To test the potential of this solution for online TE, we limited the execution of the IGP-WO method to only

TABLE II
EXECUTION TIMES OF AN OPTIMIZATION STEP
(MEDIAN VALUES OVER 3,000 RUNS)

Network	Execution time [ms]
cost266	322
geant	76
janos-us	122
nobel-germany	36

1 second, and we compared the results against the standard IGP-WO implementation, which was initialized with *Default OSPF*. As a reference, we also evaluated the results with 5-minute executions of standard IGP-WO, which represents the times reported in [8] to achieve near-optimal results in networks of similar size to those of our experiments.

Figure 2 shows the maximum link utilization achieved by the different optimization methods for samples of different topologies. We observe that by initializing IGP-WO with RBB and executing it only for 1 second it reduces by ≈30% on average the maximum link load with respect to *Default OSPF*. In general, we can see that the *RBB+IGP-WO* method significantly outperforms the standard IGP-WO method with 1-second executions. However, in samples of the *nobel-germany* topology the performance is slightly degraded. This is the smallest topology we tested. As a result, it has a narrower space of near-optimal routing configurations compared to the other topologies. This may partially explain why the algorithm performs worse here. We leave outside the scope of this paper a deeper analysis on how the topology characteristics may affect the performance of RBB.

The experimental results suggest that RBB may serve as a good initialization method for computationally-intensive TE solvers. In this case, the RBB method would only introduce a limited execution overhead at the scale of sub-seconds (see execution times in Table II).

IV. Discussion

The main difficulty in TE problems comes from the discrete nature of the cost functions. A potential limitation of RBB is that it optimizes the continuous surrogate cost, which may not approximate accurately the true discrete function. For this purpose, we control the level of approximation by tuning some parameters of our soft approximation to discrete functions (e.g., the temperature).

From a practical standpoint, RBB optimizes the link weights at the cost of executing an intensive training process. This means that it needs to spend some time to be trained before being deployed for network optimization. Once trained, the optimization time is faster than other iterative link weight optimization solutions (e.g., [7], [17]). The generalization capabilities of GNNs enable RBB to achieve a high optimization performance for a wide range of network scenarios not seen during the training process.

V. Conclusion

In this paper we presented *Routing by Backprop* (RBB), a novel method that leverages differential programming for fast TE optimization. RBB implements internally a GNN model that allows for the continuous relaxation of the optimization function (*MinMaxLoad* in our case). This eventually permits fast optimization using a gradient descent method. Our experimental results show that RBB considerably outperforms default OSPF configurations widely deployed nowadays by network operators (≈25% of improvement in our case). Moreover, we show that the proposed ML-based method generalizes successfully over different network topologies and traffic matrices not seen during training.

As future work, we plan to extend the current GNN model to support Equal Cost Multi-Path (ECMP), which is widely supported nowadays in networks running OSPF. Also, we plan to extend RBB to support the optimization of more complex performance metrics (e.g., end-to-end delay, jitter), as well as to perform multi-objective optimization.

References

[1] M. A. Lema, A. Laya, T. Mahmoodi, M. Cuevas, J. Sachs, J. Markendahl, and M. Dohler, "Business case and technology analysis for 5G low latency applications," *IEEE Access*, vol. 5, pp. 5917–5935, 2017.

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

[3] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula, "Calendaring for wide area networks," in *Proceedings of ACM SIGCOMM*, 2014, pp. 515–526.

[4] 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.

[5] 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.

[6] 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*, p. 15–28, 2015.

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

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

[9] J. Rexford, "Route optimization in IP networks," in *Handbook of Optimization in Telecommunications*. Springer, 2006, pp. 679–700.

[10] 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.

[11] "OSPF design guide," https://www.cisco.com/c/en/us/support/docs/ip/open-shortest-path-first-ospf/7039-1.html#t6, accessed: 2022-02-18.

[12] L. Liu, L. Chen, H. Xu, and H. Shao, "Automated traffic engineering in SDWAN: Beyond reinforcement learning," in *IEEE INFOCOM - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2020, pp. 430–435.

[13] R. Frostig, M. Johnson, and C. Leary, "Compiling machine learning programs via high-level tracing," 2018. [Online]. Available: https://mlsys.org/Conferences/doc/2018/146.pdf

[14] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. Mach. Learn. Res.*, vol. 18, no. 1, p. 5595–5637, Jan. 2017.

[15] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[16] 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.

[17] 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)*, 2021, pp. 1–11.