Advanced Traffic Engineering in WAN Using Graph Attention Networks

Sami Marouani, Kamal Singh, Baptiste Jeudy, Abbas Bradai, Amaury Habrard

Advanced Traffic Engineering in WAN Using Graph Attention Networks

Sami Marouani[1], Kamal Singh[1], Baptiste Jeudy[1], Abbas Bradai[2], Amaury Habrard[1,3,4]
[1]*Université Jean Monnet Saint-Étienne, CNRS, Inst. d'Optique Graduate School, Lab. Hubert Curien, F-42023 Saint-Étienne, France*
[2]*Université Côte d'Azur, CNRS, LEAT, Sophia Antipolis, France*
[3]*Institut Universitaire de France (IUF)* [4] *Inria*
Email: [1]{sami.marouani, kamal.singh, baptiste.jeudy, amaury.habrard}@univ-st-etienne.fr, [2]abbas.bradai@univ-cotedazur.fr

***Abstract*—Efficient and responsive traffic engineering is crucial for maintaining the robustness and reliability of Wide Area Networks (WANs). Traditional traffic engineering approaches often struggle to adapt to the dynamic and complex demands of today's network environments. To address these challenges, this paper enhances the Traffic Engineering algorithms by integrating an attention mechanism within the Edge-Path Embedding component. This significantly improves the model's adaptability and decision-making accuracy. Our comprehensive experimental evaluations demonstrate substantial improvements in terms of satisfied traffic demands and computational efficiency, highlighting the effectiveness of our approach.**

***Index Terms*—Attention, Graph Neural Network, Networking, Traffic Engineering, Flow Allocation**

I. Introduction

Efficient and responsive Traffic Engineering (TE) is pivotal in maintaining the robustness and reliability of WANs. As network infrastructures grow and diversify, the ability to dynamically manage vast volumes of data traffic becomes critical. Traditional traffic engineering approaches often fail to adapt swiftly to changing conditions, resulting in performance bottlenecks and underutilized network resources. This challenge is compounded by the increasing complexity of network configurations and the volatile nature of data flows across global networks. This includes inter-datacenter traffic, which requires special attention for its distinct requirements.

Recent advancements [1] have introduced machine learning (ML) models into traffic engineering, significantly enhancing predictive capabilities and operational efficiency. However, these models frequently fall short in real-time responsiveness and decision-making accuracy. This paper introduces a novel enhancement method as compared to existing graph neural network based models for WAN traffic optimization. By integrating an attention mechanism, this research aims to significantly elevate the ML based model's performance in terms of the precision and speed of TE decisions.

The main contributions of this paper are as follows:

- Attention Mechanisms: Integrating attention mechanisms in TE to dynamically prioritize critical network features.
- Performance and Adaptability: Improving satisfied demands and runtime across various network topologies.

This paper is organised as follows. Section I provides the introduction, Section II provides background on traditional traffic engineering approaches and the application of ML models in traffic engineering. Section III details the proposed graph attention-based traffic engineering model. Section IV presents the experimental setup and results, and Section V concludes the paper and discusses future work.

II. Background

Efficient TE in WANs is crucial for maintaining network performance under varying conditions. Traditional approaches often fall short and need further advancements.

A. Traditional Traffic Engineering Approaches

Traditional TE, with its reliance on static routing and predefined policies, has struggled to adapt to the dynamic demands of modern network environments. Early traffic engineering systems were not designed for the unpredictable changes in network traffic, leading to congestion and inefficient resource use. Recent advances in software-defined networking (SDN) and network function virtualization (NFV) have introduced adaptive TE solutions. SDN allows for the dynamic adjustment of traffic flows by separating the control and data planes, enhancing real-time responsiveness [2]. Similarly, NFV improves scalability and flexibility by virtualizing network services traditionally tied to specific hardware [3].

However, the full integration of adaptive TE that leverages real-time data and automated decision-making is still an open issue. Emerging applications of ML and predictive analytics in TE are beginning to allow for preemptive adjustments to traffic conditions [4]. These technologies are critical for networks that support essential applications, enhancing their adaptability and overall efficiency beyond traditional TE methods.

B. Machine Learning Models in Traffic Engineering

The integration of ML into TE marks a significant advancement toward more adaptive network operations. These models use real-time data to dynamically adjust to changes in the network environment, thereby enhancing operational efficiency [5]. For instance, Wang and Ng [6] investigated the application of ML to predict network performance in virtualized environments and improve the capability of systems to adjust routing decisions preemptively to prevent performance degradation. Recent papers, such as RouteNet-Fermi [7], have demonstrated the potential of Graph Neural Networks to accurately predict network performance metrics like delay and packet loss.

Despite these promising developments, deploying ML models in traffic engineering presents significant challenges. One major issue is ensuring the real-time responsiveness and decision-making accuracy of these models, which are critical for adapting quickly to changing network conditions. Moreover, some traditional ML models may not be well-suited for addressing networking problems, which often resemble complex combinatorial optimisation problems.

III. Graph Attention-Based Traffic Engineering

We build upon the TEAL framework [8], which leverages graph neural networks (GNNs) reinforced by deep reinforcement learning (DRL) and Alternating Direction Method of Multipliers (ADMM) for TE optimisation. Our contribution is to integrate graph attention mechanisms to better capture and adapt to diverse traffic patterns intertwined with network topologies and capacity constraints. Attention allows to selectively focus on critical nodes and edges in order to optimally allocate traffic on different paths. We show that our solution enhances traffic engineering performance by increasing satisfied traffic demands and also accelerates solution discovery times, yielding faster and more efficient decision-making.

A. Network Model and Definitions

We model the WAN traffic using a directed graph, where flows traverse specific one-way paths. Thus, the network is represented as a directed graph $G = (V, E)$ where:

- $V = \{v_1, ..., v_n\}$ represents the network nodes.
- E comprises edges with associated capacities $C(e)$, each limiting the maximum flow that can traverse the link.

Demands and Paths: Traffic demands are specified in a matrix $D \in \mathbb{R}^{n \times n}$, with each element $D_{i,j}$ indicating the data volume required from source node v_i to destination node v_j. For each demand d, P_d represents the set of all feasible paths that data can traverse from the source to the destination. $F_{d,p}$, the fraction of demand d routed through path p, quantifies the proportion of total demand d that is allocated to path p. A common TE problem is as follows [8]. Given a small number of pre-computed paths between source-destination pairs, the goal is to distribute the demands D among these paths P_d in order to maximize the overall network flow while adhering to network constraints. Specifically, we aim to maximize the total network flow, subject to network capacity constraints and the proper allocation of demands across available paths:

$$\begin{aligned} \text{maximize} \quad & \sum_{d \in D} \sum_{p \in P_d} F_{d,p} \cdot D(d) \\ \text{subject to} \quad & \sum_{p \in P_d} F_{d,p} \leq 1, \quad \forall d \in D \\ & \sum_{\substack{d \in D \\ p \ni e}} F_{d,p} \cdot D(d) \leq C(e), \quad \forall e \in E \\ & F_{d,p} \geq 0, \quad \forall d \in D, \forall p \in P_d \end{aligned} \tag{1}$$

Solvers like Gurobi [20] can find optimal solutions to such complex problem, but can be slow for large networks.

B. Graph-Based Neural Networks

Graph-Based Neural Networks have demonstrated significant effectiveness in handling graph-structured data, leading to advancements in areas such as network modeling and traffic prediction. Recent applications of GNNs encompass various domains, such as RouteNet-Fermi for network modeling and STGCN for predicting end-to-end latency [9]. Please see [10] for a detailed survey on GNN applications.

Among GNN architectures, Graph Attention Networks (GATs) [11], employ an attention mechanism to dynamically assign importance to neighboring nodes during feature aggregation. This allows GATs to focus on the most relevant nodes, thereby enhancing feature representation. In contrast, Graph Convolutional Networks (GCN) [12], extend convolutional operations to graph data by aggregating features from neighboring nodes equally to capture the graph structure.

1) Attention Mechanism in GATs: The fundamental component of Graph Attention Networks (GATs) is the attention mechanism, which evaluates the significance of the connections between each node and its neighbors. The calculation of attention coefficients, α_{ij}, determines how much influence node j has on node i. These coefficients are formulated as:

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_j]\right)\right)}{\sum_{k \in \mathcal{N}(i)} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_i \parallel \mathbf{W}\mathbf{h}_k]\right)\right)} \tag{2}$$

where $\mathbf{W}$, a learnable parameter, represents the weight matrix applied uniformly to each node's features, $\mathbf{a}$, also a learnable parameter, is the weight vector specific to the attention mechanism, $\mathbf{h}_i$ is the feature vector of node i, and $\parallel$ denotes the concatenation operation. The normalization process spans all neighbors k within node i's neighborhood, $\mathcal{N}(i)$, ensuring the sum of coefficients equals one.

2) Feature Aggregation: Following the computation of attention coefficients, the next step involves the aggregation of node features. This process updates the feature vector of each node based on the weighted contributions of its neighbors:

$$\mathbf{h}_i' = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} \mathbf{W}\mathbf{h}_j\right) \tag{3}$$

where σ represents a non-linear activation function, such as the sigmoid, used to integrate the influence from adjacent nodes.

GATs and GCNs differ in feature aggregation: GATs use attention coefficients to weigh each neighbor's importance, while GCNs treat all neighbors equally. Our research integrates GATs into the TE model, leveraging their attention mechanism to enhance the baseline. This improves upon the baseline model, demonstrating the attention mechanism's value in improving performance on network flow allocation in WANs.

C. Learning combinatorial optimisation over Graphs

Machine learning can be used to learn meta-algorithms or meta-heuristics for solving NP-hard problems [17]. Some of these works use GNN [16], [18] combined with deep reinforcement learning (DRL) [15]. The idea is that the graph embeddings can capture the situational context and graph

constraints. Next, just as a heuristic algorithm makes informed decisions to arrive at a solution, DRL based meta-heuristic can learn to make good choices by leveraging the graph embeddings. By training on multiple problem instances, DRL learns to identify crucial components in the graph embeddings that are indicative of effective problem-solving strategies. Such method uses learned patterns and relationships within the graph embeddings to guide its decision-making process. DRL's advantage is that it does not require differentiable reward functions, unlike supervised learning. This property makes it well-suited for solving optimization problems like traffic engineering, where the objective function may be non-differentiable. However, the learned algorithms may not respect different constraints. Thus, [8] added some iterations of ADMM [19] after DRL to fine-tune the solution. ADMM distributes the optimisation problem into a series of unconstrained problems by converting constraints into penalty terms in the objective function. ADMM iterations involve minimising the objective with respect to a variable at a time and then updating the variables. Thus, we use this approach by combining graph embedding with DRL and ADMM, focusing on improving graph embedding through attention mechanisms. Due to space constraints, we omit details of the DRL and ADMM phases, which are the same as in [8], and refer the reader to it.

D. Integration of Attention Mechanisms

Following the above, the architecture features a multi-tiered approach. Initially, traffic is processed using the Edge-Path Embedding stage, which is discussed later in this section. Following this, Multi-Agent RL is employed, treating each traffic demand as an individual agent. This decentralized framework enables each agent to make independent decisions while collectively striving towards a shared global objective. This setup enhances the system's capability to dynamically adjust traffic allocations in real-time. After the RL optimization phase, the ADMM is used to fine-tune these allocations, ensuring that the traffic distributions comply with network constraints such as capacity limits. These components are integrated into a cohesive architecture that balances individual traffic demands with overall network optimization goals, depicted in Figure 1.

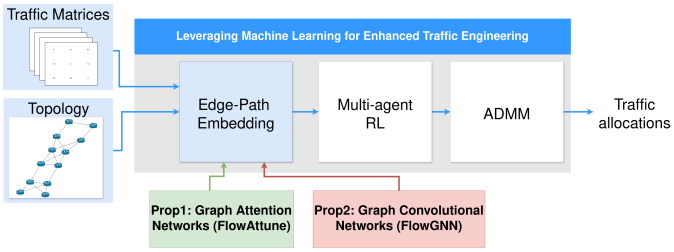

Fig. 1: Integrating attention mechanisms into Edge-Path Embedding (Green: GATs) compared to baseline architecture [8] having GCNs (Red).

Our FlowAttune model represents a significant advancement in this research, incorporating an attention mechanism into its architecture to dynamically optimize traffic across WAN. By employing this mechanism, the model effectively prioritizes and processes critical network features, thereby enhancing routing efficiency and decision-making accuracy.

FlowAttune uses six integrated layers, each comprising GAT and Dense Neural Network (DNN) components. These components operate sequentially within each layer to address network capacity and demand constraints.

GAT Operations: GATs prioritize network links by assessing their importance based on the current network conditions. Instead of representing the network topology, a bipartite graph is used. Nodes are split into two sets: one representing network edges and the other representing paths through the network. Figure 2 depicts this process, where each edge e_i influences path decisions P_j through targeted attention mechanisms. This dynamic prioritization enables the system to efficiently manage network pathways in response to real-time conditions and demands. For illustration purposes, different width of connections are shown representing different attention weights.

DNN Processing: Following the GAT operation in each layer, illustrated using black lines in Figure 2, the DNN component processes the node embeddings that result from the GAT's attention application. This step is essential for integrating the information across paths and handling the intricacies of demand constraints as well as interference due to paths crossing similar edges.

FlowAttune aims to surpass the baseline model by maximizing total feasible network flow and reducing computational runtime, improving efficiency and ensuring robust, accurate TE across WAN.

IV. Model Evaluation and Results

In this section, we evaluate the performance of our proposed model and compare it with the baseline.

A. Experimental Setup

The models were trained using an NVIDIA RTX A6000 GPU and a PC with 32GB of RAM. The topologies used were Google's B4 (12 nodes, 38 edges) from [13], and UsCarrier (158 nodes, 378 edges) and Kdl (754 nodes, 1790 edges) from Topology Zoo [14]. To train and evaluate the proposed model, we generated a comprehensive dataset of 1,036 traffic matrices for each network topology. The original 36 matrices for each topology were sourced from [8]. We generated a total of 1,000 new traffic matrices per topology from the original 36 matrices. The generation process was designed to ensure diversity and robustness in the dataset, simulating various network traffic conditions. Two primary techniques were used for generating the matrices:

- Load Adjustment: This method modified loads in existing traffic matrices by a random factor between $\pm 10\%$, using an adjustment factor selected from a uniform distribution.
- Matrix Synthesis: Generate traffic matrices by linearly combining matrices TM_A and TM_B with coefficient α, randomly chosen from a uniform distribution between 0 and 1, using the following equation:

$$\alpha \times \text{TM}_A + (1 - \alpha) \times \text{TM}_B \tag{4}$$

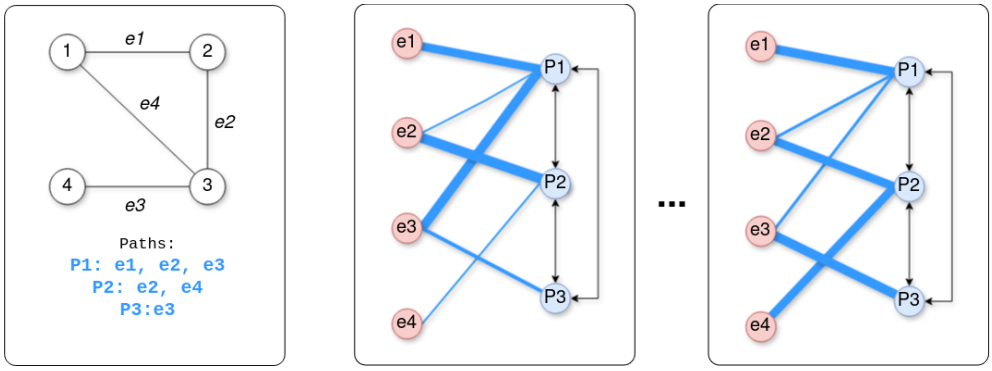

Fig. 2: Illustration of FlowAttune, with wider edges indicating higher importance to dynamically prioritizing edges.

The dataset was split into a training set (70%, 724 matrices) and validation and testing sets (15%, 156 matrices each). This division, done post-generation, ensured all sets represented the entire dataset. The maximum training epochs were set at 1000. ADMM steps were set to 2 for networks with fewer than 100 nodes and 5 for larger networks. Early stopping terminated training upon convergence before 1000 epochs.

B. Performance Analysis

During preliminary work, the baseline model showed significant performance variability. Testing the trained model with fixed traffic matrices sometimes yielded results near optimal solutions, indicating potential under favorable conditions. However, retraining and testing with the same data revealed significant deviations from the optimal, highlighting the model's lack of robustness.

To assess model reliability, we conducted ten independent training runs per topology, starting from scratch with different initial random weights. Each run took several days for larger topologies, but inference was very fast. We used Gurobi to compute optimal solutions [21] as benchmarks to evaluate how closely our model approached ideal performance.

1) Satisfied Demand Analysis: The satisfied demand indicates total feasible network flow out of total demand. We compare our enhanced FlowAttune model against baseline and optimal values using Cumulative Distribution Function (CDF) curves. These curves show the frequency of achieving certain demand levels, with closer proximity to the optimal curve demonstrating better performance. The enhanced stability and consistency of the FlowAttune model across all topologies is due to its attention mechanism, which prioritizes critical network data, streamlining decision-making and computational efforts. Figures 3, 4, and 5 show this improvement, demonstrating how the model aligns closer to optimal values and reduces result spread, ensuring reliable network management.

Table I details the performance in the UsCarrier and Kdl topologies, highlighting the impact of attention mechanisms. The satisfied demand percentages over the ten training runs show our model consistently outperforms the baseline TEAL model, except one case, nearing theoretical optimal values.

For UsCarrier, baseline satisfied demand ranges from 77.01% to 86.71%, indicating instability. The enhanced model ranges from 88.64% to 89.95%, close to the optimal 93.42%.

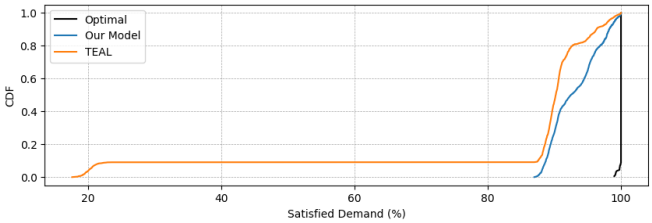

Fig. 3: CDF of Satisfied Demands for B4 Topology

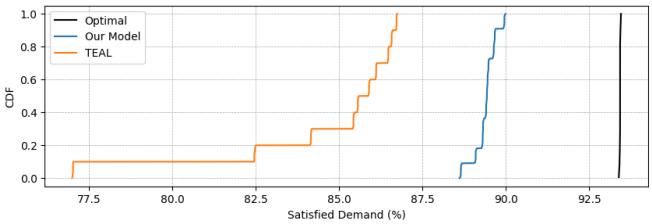

Fig. 4: CDF of Satisfied Demands for UsCarrier Topology

In Kdl, baseline results vary from 74.18% to 88.14%, showing high instability. The enhanced model achieves 79.61% to 87.23%, outperforming the baseline except one outlier. The performance also nears the optimal 95.53%. These results demonstrate the stability and performance of our model's attention mechanism for improving traffic engineering performance in complex networks.

2) Runtimes Analysis: Runtimes are a crucial performance metric in TE, reflecting the efficiency and practical applicability in real-time scenarios. This section evaluates the computational efficiency of the FlowAttune model by comparing it with both the baseline model and the optimal runtimes achieved by the optimizer across different network topologies.

For B4 topology, the enhanced FlowAttune model demonstrated a consistent improvement in runtimes compared to the

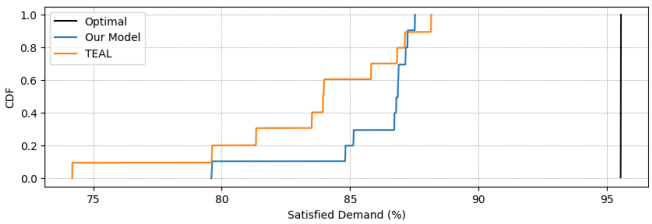

Fig. 5: CDF of Satisfied Demands for Kdl Topology

TABLE I: Training Runs and Average Satisfied Demands for Different Topologies (in percentages and in ascending order)

Independent Training Runs	UsCarrier		Kdl	
	TEAL	Our Model	TEAL	Our Model
1	77.01	88.64	74.18	79.61
2	82.46	89.09	79.61	84.81
3	84.15	89.31	81.33	85.12
4	85.43	89.40	83.50	86.71
5	85.56	89.42	83.93	86.79
6	85.89	89.44	83.97	86.85
7	86.10	89.47	85.81	86.88
8	86.47	89.63	86.82	87.15
9	86.57	89.66	87.12	87.23
10	86.71	89.95	88.14	87.51
Avg. for Gurobi	93.42		95.53	

baseline TEAL model. Figure 6 shows that the FlowAttune model achieved runtimes between 4.39ms and 31.67ms, consistently showing faster performance than the TEAL model, which recorded runtimes ranging from 7.449ms to 32.59ms. This indicates that FlowAttune typically operates faster than TEAL in most cases. Although the optimizer achieved the fastest runtimes, approximately from 1.118ms to 2.757ms, the relatively small size of this topology does not necessitate extensive computational time to find optimal solutions.

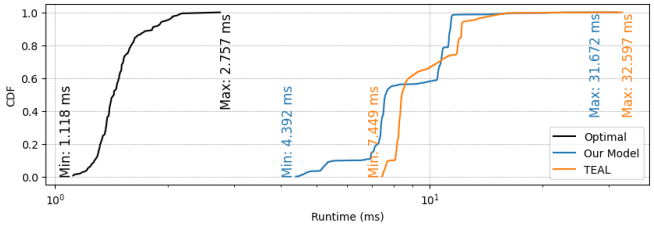

Fig. 6: CDF of Runtime for B4 Topology.

For UsCarrier, the enhanced model achieved runtimes ranging from 26.14ms to 183.88ms, as shown in Figure 7. In comparison, the baseline TEAL model's runtimes varied from 37.64ms to 324.85ms, indicating that our model typically operates faster, also depicted in the runtime distribution. The optimizer required significantly more time from 6889.313ms to 1014.468ms, to find the optimal solution, due to increased computational demand as the network's complexity grows.

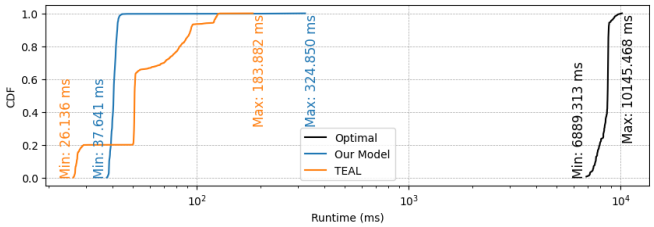

Fig. 7: CDF of Runtime for UsCarrier Topology.

In Kdl, the FlowAttune demonstrated significant efficiency, with runtimes between 0.92s and 1.74s. The baseline TEAL model's runtimes were more variable, ranging from 1.70s to 2.72s. The runtime comparison shown in Figure 8 illustrates that our model consistently outperforms TEAL, effectively reducing the computational time required for processing. The optimizer's runtimes in this topology extended from 1715.25s to 2217.57s, further emphasizing the computational intensity needed for finding optimal solutions in larger, complex networks. A summary of average runtimes for these topologies across all the training runs can be found in Table II.

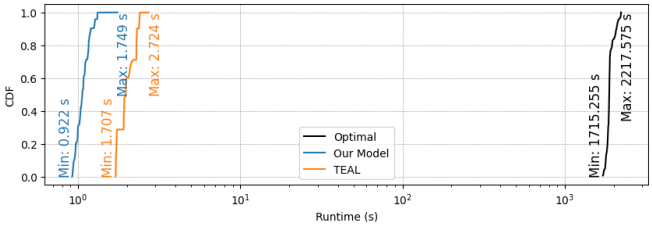

Fig. 8: CDF of Runtime for Kdl topology

TABLE II: Training Runs and Avg. Runtimes for UsCarrier and Kdl Topologies (in ms, sorted in ascending order)

Independent Training Runs	UsCarrier		Kdl	
	TEAL	Our Model	TEAL	Our Model
1	27.12	38.68	1716.1	934.4
2	28.44	38.88	1723.2	965.3
3	50.56	40.10	1738.4	997.5
4	50.80	40.18	1917.0	1032.1
5	50.92	40.88	1934.3	1057.8
6	51.10	41.53	1975.0	1083.3
7	82.44	41.88	2101.1	1112.0
8	83.65	42.81	2292.7	1158.2
9	84.15	43.29	2306.0	1181.2
10	85.07	43.40	2393.9	1291.7
Avg. for Gurobi	8455.1		1900193	

The attention mechanism within the FlowAttune model plays a pivotal role in these performance gains. By effectively prioritizing and processing the most relevant features of the network data, the model minimizes unnecessary computations. This not only accelerates runtime but also ensures optimal use of computational resources, which is especially beneficial in large-scale and complex network topologies.

C. Overall comparison

Table III highlights the performance gains achieved by our model compared to the Baseline across several network topologies. In the B4 topology, our model improved the satisfied demands by 10.2% while reducing the runtime by 8.33%, demonstrating enhanced efficiency and faster processing. For the UsCarrier topology, the improvements are equally compelling, with a 5.62% increase in total transmitted demands and a substantial 30.80% reduction in runtime, underscoring our model's capability to manage larger and more complex network environments effectively. In the Kdl topology, although the improvement in satisfied demands is more modest at 2.89%, the reduction in runtime is remarkable at 46.36%.

The standard deviation (*std*) of satisfied demands provides insights into the model's consistency across different runs. Lower std values indicate more stable performance, while

higher std values suggest greater variability. Notably, our model shows a marked improvement in stability. In the B4 topology, std reduced from 21.61 for the Baseline to 3.62, indicating enhanced performance consistency. For the UsCarrier topology, std improved from 2.82 to 0.33, and for the Kdl topology, it improved from 3.95 to 2.28. These reductions underscore the reliability and enhanced stability of our model across varying network conditions.

TABLE III: Performance metrics and improvement comparisons across different topologies for Baseline, Gurobi, and our model, presenting satisfied demand (%) and runtime (ms) as mean $\pm$ standard deviation.

Metric	B4	UsCarrier	Kdl
Satisfied D. (%)			
Baseline	84.40 ± 21.61	84.64 ± 2.82	83.44 ± 3.95
Our Model	93.05 ± 3.62	89.40 ± 0.33	85.85 ± 2.28
Gurobi	99.96 ± 0.15	93.42 ± 0.01	95.53 ± 0.00
Runtime (ms)			
Baseline	9.6 ± 2.1	59.4 ± 25.9	2008.7 ± 238.8
Our Model	8.8 ± 2.3	41.1 ± 11.4	1077.4 ± 105.1
Gurobi	1.4 ± 0.2	8455.0 ± 525.2	1900193 ± 108862
Improv. over Baseline			
Satisfied D. (%)	10.2	5.62	2.89
Runtime (%)	8.33	30.80	46.36
Improv. over Baseline & Gurobi			
Satisfied D. (%)	55.57	54.21	19.93

To assess the performance improvements of our model relative to the Baseline and the optimal results, we employ a normalized performance metric defined as:

$$\text{Norm. Satisfied D. } (\%) = 100 \times \frac{V_{\text{Our Model}} - V_{\text{Baseline}}}{V_{\text{Optimal}} - V_{\text{Baseline}}} \quad (5)$$

where V represents the total transmitted demands.

Applying this formula, we observe the following:

- **B4 Topology**: The normalized satisfied demands of our model is 55.57%, indicating that it achieves more than half of the possible improvement between the Baseline and the optimal value.
- **UsCarrier Topology**: With a normalized satisfied demands of 54.21%, our model shows that it captures more than half of the potential improvement.
- **Kdl Topology**: The normalized satisfied demands is 19.93%, demonstrating that our model achieves nearly a fifth of the way from the Baseline to the optimal value.

FlowAttune significantly improved runtime over the baseline across all topologies, confirming its suitability for real-time TE. Its efficiency is especially notable in larger networks, where traditional methods become computationally intensive, making it ideal for quick decision-making and high scalability.

V. Conclusion

The integration of attention mechanisms into the Edge-Path Embedding component has demonstrated substantial improvements in both satisfied demands and runtime across various network topologies. The enhanced FlowAttune model mostly outperforms the baseline, showing greater stability and robustness, particularly in larger and more complex network environments. Future work will explore further refinements of the attention mechanism and its application to additional network scenarios to continue improving WAN TE.

Acknowledgment

This work has been supported by grant ANR-21-CE25-0005 from the Agence Nationale de la Recherche, France for the SAFE project.

References

[1] Guillermo B. "Is machine learning ready for traffic engineering optimization?." in *IEEE Int. Conf. on Network Protocols (ICNP)*, 2021.

[2] Kreutz, Diego, et al. "Software-defined networking: A comprehensive survey." *Proceedings of the IEEE* 103.1 (2014): 14-76.

[3] Mijumbi, Rashid, et al. "Network function virtualization: State-of-the-art and research challenges." *IEEE Communications Surveys & Tutorials* 18.1 (2015): 236-262.

[4] Boutaba, Raouf, et al. "A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 16, 2018.

[5] Minghao Ye, et al. "FlexDATE: Flexible and Disturbance-Aware Traffic Engineering With Reinforcement Learning in Software-Defined Networks." *IEEE Trans. on Networking*, vol. 31.4, pp. 1433-1448, 2023.

[6] Wang, J. and Ng, T. S. E. "The impact of virtualization on network performance of Amazon EC2 data center," in *IEEE INFOCOM*, 2012.

[7] Galmés, Miquel Ferriol, et al. "RouteNet-Fermi: Network Modeling with Graph Neural Networks," in *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 6, pp. 1701-1711, 2021.

[8] Xu, Zhiying, et al. "Teal: Learning-Accelerated Optimization of WAN Traffic Engineering," in *ACM SIGCOMM*, pp. 378-393, 2023.

[9] Ge, Zhun, Jiacheng Hou, and Amiya Nayak. "Forecasting SDN end-to-end latency using graph neural network." *2023 International Conference on Information Networking* (IEEE, 2023).

[10] Zhou, Jie, et al. "Graph Neural Networks: A Review of Methods and Applications," in *AI Open*, vol. 1, pp. 57-81, 2020.

[11] Veličković, Petar, et al. "Graph Attention Networks," *arXiv preprint arXiv:1710.10903*, 2018.

[12] Kipf, Thomas N., and Welling, Max, "Semi-Supervised Classification with Graph Convolutional Networks," *arXiv preprint arXiv:1609.02907*, 2016.

[13] Jain, Sushant, et al. "B4: Experience with a globally-deployed software defined WAN." *ACM SIGCOMM Computer Communication Review* 43.4 (2013): 3-14.

[14] Knight, Simon, et al. "The internet topology zoo." *IEEE Journal on Selected Areas in Communications* 29.9 (2011): 1765-1775.

[15] Khalil, Elias, et al. "Learning combinatorial optimization algorithms over graphs." *Advances in Neural Information Processing Systems* 30 (2017).

[16] Peng, Yun, Byron Choi, and Jianliang Xu. "Graph learning for combinatorial optimization: a survey of state-of-the-art." *Data Science and Engineering* 6.2 (2021): 119-141.

[17] Nair, Vinod, et al. "Solving mixed integer programs using neural networks." *arXiv preprint arXiv:2012.13349* (2020).

[18] Gasse, Maxime, et al. "Exact combinatorial optimization with graph convolutional neural networks." *Advances in Neural Information Processing Systems* 32 (2019).

[19] Boyd, Stephen, et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers." *Foundations and Trends in Machine Learning* 3.1 (2011): 1-122.

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

[21] Abuzaid, Firas, et al. "Contracting wide-area network topologies to solve flow problems quickly." *18th USENIX Symposium on Networked Systems Design and Implementation* (NSDI 21). 2021.