# Topology Aware Deep Learning for Wireless Network Optimization

Shuai Zhang, *Student Member, IEEE*, Bo Yin, *Student Member, IEEE*, Weiyi Zhang, *Senior Member, IEEE*, and Yu Cheng, *Senior Member, IEEE*

***Abstract*— Data-driven machine learning approaches have been proposed to facilitate wireless network optimization by learning latent knowledge from historical optimization instances. However, existing works use simplistic network representations that cannot properly encode the topological difference. They are often limited to fixed topology, and the performance is degraded because the learning target does not get sufficient information since the topological information is not well captured. To address this, we leverage the graphical neural network techniques and propose a two-stage topology-aware deep learning (TADL) framework, which trains a graph embedding unit and a link usage prediction module jointly to discover links likely to be used in optimal scheduling. By properly encoding the network structure, it makes input data with varying topology possible, and also provides more informative clues for the learning target. Important techniques are developed to ensure learning efficiency. The performance is evaluated on canonical multi-hop flow problems with diverse network structures, sizes and realistic deployment scenarios. It achieves close-to-optimum solution quality with a significant reduction in computation time without retraining.**

***Index Terms*— Deep learning, wireless network optimization, topology representation.**

## I. Introduction

OVER decades, wireless networking has become an indispensable part of today's society. The recent proliferation of the Internet-of-Things (IoT) and the dense deployment of next-generation wireless network access points or base stations have greatly increased the scale and complexity of wireless networks. The study of wireless network optimization, although legacy, still plays a key role in modern wireless networks; with the pressure of large scale high complexity, and more dynamics, the need for efficient computation algorithms that can optimize network resource allocation in an adaptive and timely manner is more urgent.

Most network optimization tasks follow the paradigm of mathematical programming: given the constraints of resource budget or exclusive usage at a single time, performance metrics or system utility can be improved through judicious allocation and scheduling. Specifically, with the broadcast nature of wireless networks, at any given time, only a subset of communication links can be activated concurrently to mitigate interference [1]–[3] while achieving a high system performance figure such as throughput. With such a combinatorial interference structure, it is common that most performance optimizations in wireless networks are NP-hard [4]. To tackle this challenge, studies on wireless network optimization in the past years focused on the development of approximation algorithms [5]–[7], based on the mathematical model of the performance goal and the constraints. One drawback of this traditional model-based optimization approach is that the computation *experience* gained from solving historical problem instances is wasted: whenever changes to the networking setting occur, (for example, user traffic pattern can drift with time, the set of active users may change due to mobility, or network topology can shift due to the adjustment of resources), the optimization procedure needs to be rerun [8], and there is no difference in the process even if the new situation is almost identical to the previous network condition.

Inspired by the recent breakthroughs in machine learning (ML), data-driven approaches receive much attention in the study of wireless network optimization, especially in vehicular networks, which is closely linked to the classic yet highly practical problem of autonomous driving [9], [10]. One major thread of exploiting the historical data for network management adopts the methodology of reinforcement learning (RL), and especially the experience replay technique, allows an agent to learn a reasonable control policy from its past interactions with the environment. Research attempts along this line have developed RL-based algorithms to address a large range of network optimization problems, including access control [11], [12], network scheduling [13], [14], and traffic engineering [15], [16]. These studies typically focus on the optimization tasks for a specific layer (physical layer, link layer, or network layer) or simple single-hop networking scenarios, where the target system can be conveniently modeled by a Markov decision process (MDP), and the application of ML in the cross-layer optimization in wireless networks is often limited.

Works that follow a supervised-learning approach utilize historical problem instances that are solved by conventional algorithms as training data, from which the trained machine can learn a mapping function to predict or facilitate computing the solution of a new problem instance [17]–[20], where the

Manuscript received 14 October 2021; revised 23 March 2022; accepted 16 May 2022. Date of publication 8 June 2022; date of current version 11 November 2022. This work was supported in part by NSF under Grant CNS-1816908 and Grant CNS-2008092. The associate editor coordinating the review of this article and approving it for publication was X. Cheng. *(Corresponding author: Yu Cheng.)*

Shuai Zhang, Bo Yin, and Yu Cheng are with the Electrical and Computer Engineering Department, Illinois Institute of Technology, Chicago, IL 60616 USA (e-mail: cheng@iit.edu).

Weiyi Zhang is with AT&T Labs Research, Middletown, NJ 45697 USA.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TWC.2022.3179352.

Digital Object Identifier 10.1109/TWC.2022.3179352

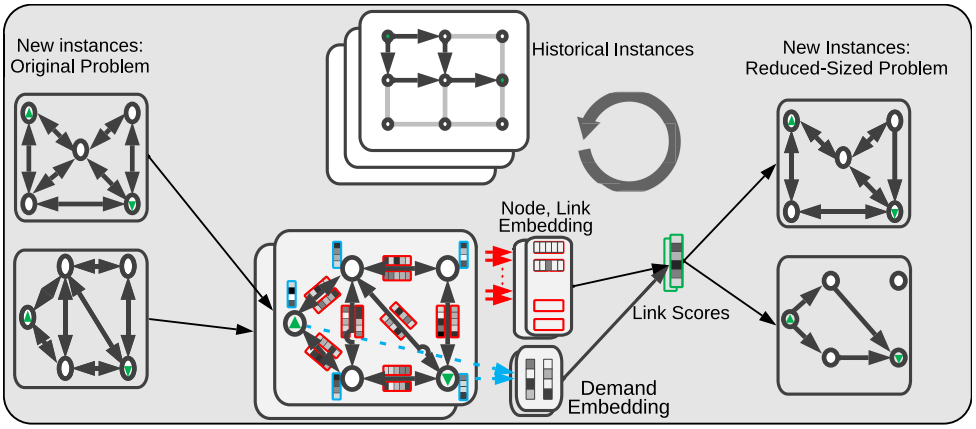



Fig. 1. The proposed topology-aware deep learning (TADL) framework. The trained model would convert input data of various topology into representing vectors, which are used for predicting the usefulness of links to obtain a sub-network of likely links.

new instances do not change the topology or the applications are only concerned with single-hop scenarios. However, it was not clear how such supervised learning based approaches could be adopted for optimization over multi-hop wireless networks. The fundamental obstacle is the lack of appropriate presentation of multi-hop wireless network optimization in the ML framework, which is rooted in the concept of classification.

As a result of these considerations, in this paper, we focus on the problem of network flow optimization over a generic multi-hop wireless network with a supervised learning approach. Such research issues had been one key area in the recent two decades [21], [22], and are expected to still play an important role in emerging areas such as vehicular ad-hoc networks and aerial access networks. Our preliminary studies in [8], [23] for the first time, to the best of our knowledge, contributed a method to integrate supervised deep learning with multi-hop wireless network flow optimization. In [8], [23], the optimization solutions from historical problem instances are leveraged as training data to conduct supervised learning. The innovative angle is to transform the optimal flow allocation over each link into a normalized importance index, which paves the way for conducting learning in the classification-based framework. After training, the machine gains the capability to predict the usage likelihood of each link, given a new problem instance; links with low predicted values are then pruned off from the problem so that the effective optimization problem scale is reduced with only minor solution quality degradation.

These preliminary studies are, however, limited to the situation with a given static topology. In this paper, we aim to take up the challenge of extending the methodology to a more generic scenario with dynamic topologies. Effective re-optimization over various topologies will be particularly useful if a wireless network needs to timely handle the dynamics due to user mobility, traffic pattern change, or adjustment of network resources. A straightforward idea to incorporate topology into learning is feeding the topology information, in the form of the adjacency matrix, directly to the machine. However, the topology representation based on the adjacency matrix will be dependent on the specific node indexes. With dynamic networks, the communication nodes may acquire new indexes due to mobility or update of resource allocation, and the situation of the same topologies but with different node indexes will not be rare. The fundamental difficulty is there is no canonical representation for graph data, and to tell if two graphs representations are structurally equivalent is shown to be computationally difficult [24].

So far, graphical neural networks (GNN) provide new ways to address the challenge of representing graphical problem data in a form that facilitates training and inference. Instead of straightforwardly treating the input graph data as mere vectors, GNN provides a way to represent the structure of a graph and specified interactions among the elements within the graph in an order-invariant manner.

Leveraging on that, we propose a topology-aware deep learning (TADL) framework as illustrated in fig. 1 for wireless network flow optimization. This is to tackle the aforementioned drawback of current scheme not properly handling dynamic topologies. In TADL, graph embedding techniques are developed to incorporate structure-level topological information so that its output representations have network structure information built into it which is independent of node or link indexes. We show that by learning on such a topology-aware representation of the problem data, neural networks can accurately infer the links to be used (and thus prune non-critical links), leading to a good trade-off between problem scale reduction and closeness to an optimal solution, robustly over various network topologies and different commodity flow deployment scenarios, for example, on average maintaining at least 88% of the optimum network throughput while lowering the computation time cost by over 60%, across a wide range of network topology from network scales up to 200 nodes.

In summary, this paper incorporates three-fold innovative contributions:

- We design the TADL framework that integrates graph embedding, attention mechanism, and specially tailored implementation techniques. When appropriately trained with a sufficient amount of data, TADL achieves robust link usage predication over various network topologies and different commodity flow deployment scenarios without retraining.

- Our embedding methods incorporate node vector, link vector, and interactions among those vectors through parameters determined by training, and pose no restrictions on the network size to be processed. It is such a design that enables the embedding to encode the impact of a certain link on the end-to-end (i.e. commodity flow level) performance under complex interference relationships.
- We develop important learning techniques which are essential to ensure good performance in topology-aware learning, including the proper design of loss function, sample selection with curriculum training, and feasibility guarantee with link pruning.
- We evaluate the topology-aware extensibility of the proposed model with a new quantitative measure and present extensive numerical results with insights including the new dimension of complexity-performance tradeoff enabled by TADL, the impact of the training data volume on the learning performance in the context of wireless network optimization, and the applicability of the learned capability in new settings without retraining.

The remainder of this paper is organized as follows. Section III concisely describes the wireless network optimization problem and associated conventional algorithm, which is to be studied with a machine learning approach in this paper. Section IV presents the proposed TADL framework and related implementation details. Numerical results and performance evaluations are presented in Section V. Section II reviews more related work. Section VI concludes the paper.

## II. Related Work

### A. Conventional Wireless Network Optimization

Wireless network optimization had been a key research area in the recent two decades. The basic methodology is to compute the resource allocation aspects such as channel assignment, base station association, scheduling, and power control using various mathematical programming algorithms [25]. Due to the complex inference relationship, wireless network optimization is NP-hard in general, and the major thread of efforts in the community is the development of various approximation algorithms [1], [26]. The studies had also been extended from single-radio single-channel context to complex multi-radio multi-channel context [27]–[29]. A particular issue inspiring the machine learning study in [8] and this paper is that a new optimization problem instance is always solved either from scratch or with a trivial re-optimization approach [30]; machine learning aims to exploit the historical computation effort to benefit new optimization instances.

### B. Data-Driven Solutions to Network Problems

Aspects of network design problems can be cast as optimum control problems, and there have been attempts to apply machine learning methods [31], [32] as a way to discover heuristic algorithms from data.

For problems with a need for a sequential decision process, deep reinforcement is used instead. Many of them have a problem scenario that can be cast as a Markovian Decision Process, and the solution in each step depends only on the system's current state. This line of research is shown by a series of online network control problems [16], [33], [34]. A particularly relevant work [16] uses an advanced deep reinforcement learning actor-critic framework to produce the optimum routing path in a data-center network.

The majority of the work uses a supervised learning approach and sometimes assisted by unsupervised learning as a pre-training step. The straightforward approaches [19], [35] treat machine learning as a black-box approximator for producing approximate output to replace an existing computation block. As a typical example, deep learning is used to approximate the weighted minimum-squared error power allocation solution to maximize the network throughput [19]. Our previous work [8] typifies the supervised approach to optimize the flow scheduling in wireless ad-hoc networks, with improvements from unsupervised pretraining. Later works focus on bringing structure information into the model: for example, a recent work uses graph embedding technique is used for power control in a D2D network [36]. The embedding block is used to generate an embedding vector for each D2D pair, and the learning results in a mapping from such vectors to the power decision based on them.

Our work is set apart from the existing approaches in important ways. In terms of the problem itself, though there are works with a similar D2D network setting, we propose the neural network solution to multi-hop routing under complex link interference relationships, while others study simpler settings, such as node random access strategy, which can be efficiently learned without the specific link-level information.

Our proposed TADL solution differs from other GNN-integrated solutions. It employs both node and link embedding vectors, while others focus more on the information propagation of either nodes or links. And the use of attention mechanism makes the model output and input not be limited to a fixed number of input or output, which gives an additional layer of flexibility. As a result, it handles changing topologies without assuming artificial limits like the number of nodes or links, and integrates information both from nodes and links for making a better quality decision.

## III. System Model

In this paper, we consider a multi-commodity flow (MCF) problem as a concrete context to demonstrate our method. We give a brief introduction to the basics of the MCF problem in this section. Note that although the problem is presented in a single-radio single-channel (SRSC) setting, a generic multi-radio multi-channel (MRMC) wireless network can be mapped as a virtual SRSC with the multidimensional tuple modeling technique developed in [8], [23], [37], [38].

### A. Network Model

The SRSC network is represented by a directed graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$, where $\mathcal{N}$ and $\mathcal{E}$ denote the sets of nodes and links, respectively. A communication link $e \in \mathcal{E}$ exists from node $u$ to node $v$, denoted by tuple $(u, v)$ if node $v$ is within the *communication range* of node $u$. Each link has a physical

TABLE I
THE LIST OF MATHEMATICAL SYMBOLS USED IN THE PROBLEM STATEMENT

| Symbol | Definition |
|---|---|
| $\alpha_m$ | time fraction assigned to IS $m$ |
| $\mathcal{C}$ | set of commodity flow demands |
| $c(u,v)$ | capacity of link $(u,v)$ |
| $f_k(u,v)$ | flow of commodity $k$ on link $(u,v)$ |
| $K$ | number of flow demands |
| $\mathcal{M}$ | set of all ISs |
| $\mathcal{N}$, $\mathcal{E}$ | set of nodes, set of links |
| $r_k$ | the achievable throughput of commodity $k$ |
| $s_k, d_k$ | the source and destination nodes of demand $k$ |

transmission capacity $c(u,v)$, specifying the peak data rate this link can support.

We consider the protocol interference model [39]. A link can transmit, or become *activated*, only when no other transmitting node belonging to a different link is within the *interference range* of its receiving node. The scheduling is also under a radio constraint such that links sharing the same node cannot be activated simultaneously. As a result of these constraints, the conflict relations among all links in the network can be characterized by a conflict graph [1], whose vertices are network links and there is an undirected edge if two links interfere. Thus, an independent set (IS) over the conflict graph indicates a set of links that can be scheduled for transmission simultaneously.

In the network, there exist $K$ commodity flow demands, denoted as $\mathcal{C}$. A commodity flow $k$ can be represented by the pair $(s_k, d_k)$, with $s_k$ being the source node with a positive net outward data flow and $t_k$ the sink node with a positive net inward flow. Let $f_k(u,v)$ denote the amount of traffic flow associated with commodity $k$ on link $(u,v)$. For commodity $k$, the achievable throughput $r_k$ is the net flow out of the source node as

$$r_k = \sum_{v \in \mathcal{N}_{s_k}^+} f_k(s_k, v) = \sum_{v \in \mathcal{N}_{d_k}^-} f_k(v, d_k), \tag{1}$$

where $\mathcal{N}_v^-$ (resp. $\mathcal{N}_v^+$) denotes the set of in-neighbors (resp. out-neighbors) of $v$.

### B. Problem Statement

We here consider the network flow problem of maximizing the minimum commodity flow in the network. In a wireless setting, a network flow problem involves not only routing decisions but also scheduling decisions in which ISs are activated in a time-multiplexing manner. Besides the optimization variables $f_k(u,v)$ that specify the routing decisions, let $\mathcal{M}$ be the set of all ISs and $\alpha_m$ denotes the fraction of time scheduled to IS $m$. Those decision variables need to satisfy the following constraints.

*1) Link Capacity Constraints:* The sum of all the flows over a link does not exceed its capacity across all the activated time periods, i.e.,

$$\sum_{k=1}^{K} f_k(u,v) \le \sum_{m \in \mathcal{M}} \alpha_m p_m(u,v), \quad \forall (u,v) \in \mathcal{E}, \tag{2}$$

$$p_m(u,v) \triangleq c(u,v) \quad \text{if} (u,v) \text{ is active in } m. \tag{3}$$

$p_m(u,v)$ is the effective capacity of link $(u,v)$ in an IS $m$: its value is $c(u,v)$ if link $(u,v)$ is activated in $m$, and 0 otherwise.

*2) Flow Conservation Constraints:* For any commodity flow $k$, the amount of flow entering an intermediate node equals to that exits the node, i.e.,

$$\sum_{u \in \mathcal{N}_v^-} f_k(u,v) = \sum_{u \in \mathcal{N}_v^+} f_k(v,u), \quad \forall v \neq s_k, d_k; \forall k. \tag{4}$$

*3) Scheduling Time Constraint:* The time fraction assigned to all ISs must sum to 1, as

$$\sum_{m \in \mathcal{M}} \alpha_m = 1. \tag{5}$$

With the minimum of all the achieved commodity flows denoted as $z$, the MCF problem can be formulated as follows.

$$\begin{aligned} \underset{\{f_k(u,v)\},\{a_m\}}{\text{Maximize}} \quad & z \\ \text{s.t.} \quad & \text{constraints}(2),(4),(5), \\ & z \le r_k, \quad \forall k \\ & f_k(u,v) \ge 0, \quad \forall (u,v) \in \mathcal{E}, k; \\ & \alpha_m \ge 0, \quad m \in \mathcal{M} \end{aligned} \tag{6}$$

The problem described above has the form of linear programming because the objective and constraints are linear functions. However, the size of $\mathcal{M}$ is exponentially large and cannot be easily enumerated; even to obtain one set of non-interfering links is equivalent to finding a graph coloring of links. Therefore the problem is essentially a mixed-integer linear programming (MINLP) type with an exponential number of variables.

### C. Delayed Column Generation Method

To the best of our knowledge, the most efficient approximation with guaranteed bound analysis is the delayed column generation (DCG) method [40], [41]. Starting from an initial set of ISs, it solves a series of partial problems and uses the dual solution to generate new ISs to add to the problem. Following this procedural column generation, a solution sufficiently close to the optimum solution to the original problem is obtained. In this way, the algorithm memory usage is saved and complexity can be controlled as a trade-off with the objective. The reader is referred to the work [41] for a more detailed account of this method. In the remainder of this paper, we will use the DCG method to solve sample optimization instances. The DCG solutions will be used as training data for supervised learning, and serve as performance bench mark to evaluate the solution quality when our proposed machine learning technique is applied. For the convenience of presentation, we term the DCG based solutions as "optimal solutions."

*Remark:* We take the multi-commodity flow over a multi-hop wireless network as specified in this section under study with two considerations: 1) the independent-set-based scheduling, NP-hard in general, has been one of the key challenges in wireless network optimization; 2) this problem formulation provides a suitable concrete setting to conduct the topology-aware machine learning targeted in this paper. However, the benefits of the proposed learning-based method and related principles and insights generated from this paper are by no means applicable to this problem only.

## IV. Topology-Aware Deep Learning Framework

The proposed topology-aware deep learning (TADL) framework is shown in fig. 1, which illustrates the whole procedure of supervised learning, the application of learning algorithm for link prediction, and the neural network structure enabling the topology-aware capability. Specifically, the machine is equipped with node and link embedding units, a demand embedding unit, and an attention unit. The embedding units are mainly used to address the issues of topology representation as highlighted in the introduction. They learn a reasonable representation of a problem instance, including multiple link embedding vectors and a demand embedding vector. Such a representation is leveraged by the attention unit to identify network links that are likely to be used in an optimized way.

The remainder of the paper adopts some common mathematical notations. We use a boldface uppercase/lowercase letter to represent a matrix/vector, respectively. We use $\|$ to express vector concatenation and $\odot$ for element-wise multiplication. LIN represents a parameterized linear layer $\mathbf{y}(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$; MLP denotes a multi-layer perceptron [42], made up of several dense neural network layers, each with individual non-linear activation function. $\mathbf{D}$ and $\mathbf{A}$ are degree and adjacency matrix following the conventional graph theory definitions.

### A. Principles of Topology-Aware Embedding

*1) The Challenge of Topology Representation:* We consider the topology-aware vector representation to be an indispensable part of applying machine learning to network research, because the lack of it leads to undesirable fitting on the network element ordering. Assume that one directly learns from the usual representations like adjacency matrix $\mathbf{A}$, where an element $\mathbf{A}[i, j]$ is 1 if there is a connection between node $i$ and $j$ and 0 otherwise, it would be difficult to correctly represent the structure for the following two reasons.

One concern is that the final decision is likely to depend on the specific order of the elements. Because from the perspective of learning algorithms, permuted adjacency matrices are treated as vastly different inputs. If the permuted version does not appear in the training data, it is unlikely that they both correspond to the same output.

The order-related issue is illustrated with an example in fig. 2. Network $G$ is given with its nodes numbered in the clockwise order, and network $G'$ is obtained from $G$ by rearranging its nodes.[1] We use a neural network model to predict the set of links to be used in the optimum scheduling. The model is an enhanced version of [8] by explicitly including the adjacency matrix as input to the machine, denoted as method ADJ in section V. The ADJ method is applied for link prediction in $G$ and $G'$, when one commodity flow is deployed and other configurations follow that given in Section 4. The results in these two cases should be identical, because the network topology and the nodes with traffic demands are identical. However, two different link sets are predicted for $G$ and $G'$, shown by the different sets of red links in the two cases. The reason behind this is that neural networks with no topology-awareness are prone to treat the superficial difference in the adjacency matrices, caused by element order, as fundamentally different inputs, as shown in Figure 2. Section 4 presents more numerical results demonstrating the inefficiency of the ADJ method.

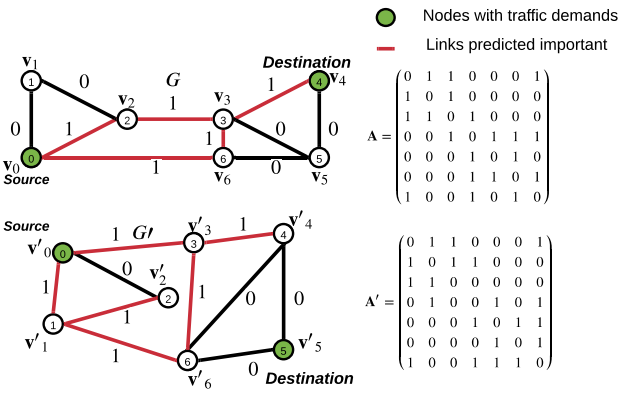



Fig. 2. A learning framework directly using the adjacency matrix leads to different link predictions.

The other concern is that the adjacency matrix representation will be sparse for typical networks of non-trivial sizes since the number of usable links $|\mathcal{E}|$ is far below the number of possible node pairs $|\mathcal{N}|(|\mathcal{N}-1|)$. Learning from sparse data is difficult to do with today's learning frameworks [43].

From these arguments, we see that the intrinsic difficulty is due to the fact that graphical structures lack an inherent order: unlike pictures or time series where there is a spatial or temporal ordering by which the representation can be made unique, for networks, the nodes and edges information could be passed in any order to the learning algorithm while not changing the underlying mathematical object. But neural networks are exceedingly capable of picking up *any* pattern in the data; these patterns could arise from the particular ordering of the element, which is undesirable because the problem's solutions are not a function of that particular order. While this serves well in other applications like image classification, it hurts the generalization ability in network-based problems [44]. On the other hand, generating a representation which corresponds to the graph structure only without being affected by the element ordering is reducible to the Graph Isomorphism problem in

[1] Intuitively, node 1 in $G$ is dragged into the area surrounded by nodes 0, 2, 3, 5, and the topology is then flipped vertically; and the nodes are renumbered in the clockwise order. The nodes $(0, 1, 2, 3, 4, 5, 6)$ in $G$ have a one-to-one correspondence to nodes $(0, 2, 1, 6, 5, 4, 3)$ in $G'$.

graph theory. Its complexity is non-trivial and still yet to be confirmed to belong to the NP-class [45] or not.

Considering the importance and practical difficulty of characterizing network problems based on its topology as discussed above, we tackle the issue by developing embedding techniques that have the following properties:

1) The output value shall be constant with respect to the changes in the order of network nodes or links. This implies that we get the same evaluation of link importance regardless of how the node or edge changes its index.
2) The output value shall be able to handle various and changing network scales. This requires that when generating the vector representations of the problem instance, there should not be any assumption of network size.
3) The vector representation of nodes or links should directly encode associated feature information. This item is necessary as in network applications, the associated information (e.g., node queue capacity, link strength) is as important as the graph structure itself in determining the output.

The end result is a network topology-based "signature" based on each graph element, such that the learning algorithm could have access to not only the node or link information, but also how they interact with each other and form a whole network.

*2) Index-Independent Embedding:* To achieve the goals stated above, we use the message-passing network [46] approach. It iteratively updates the link and node embedding vectors based both on the feature vectors and the graph topology. This process is also known in other works as the graph convolutional network [47]. It works in a way that is parallel to how a network operates in the normal working state: each node sends to and receives from its neighbors and makes updates, and as a result, the updated states are an implicit function of graph topology. After a few rounds, every node has partial information about the neighborhood it is in. In this way, the neural network computes a vector representation for each node and links in the network that can accommodate the changing network topology and scale.

We then embed the network's traffic demand information by the node and link embedding vectors, (section IV-B),, and use the embedded traffic demand and embedded link vectors to obtain the output (section IV-C).

Our proposed embedding technique can be intuitively expressed in eq. (7):

$$\begin{aligned} \mathbf{E}_{\text{emb}}, \mathbf{V}_{\text{emb}} &= \text{NN}_1(\mathbf{E}, \mathbf{V}, \mathcal{G}) \\ \text{solution} &= \text{NN}_2(\text{NN}_3(\mathcal{C}, \mathbf{V}_{\text{emb}}), \mathbf{E}_{\text{emb}}). \end{aligned} \tag{7}$$

$\text{NN}_1$ uses GNN to process as input the network graph information and outputs embedded node and link vectors. It corresponds to the operations defined in section IV-B. $\text{NN}_2$ uses the node and link vectors generated by $\text{NN}_1$ to predict the link importance in the scheduling task, described by eq. (17). $\text{NN}_3$ represents the module used to process the network flow demand information. It uses the vectors generated by $\text{NN}_1$, and it corresponds to the process described by eq. (16). Note that notations $\text{NN}_{1,2,3}$ are only abstractions of the components of our proposed method, and their corresponding definitions are defined in later subsections.

### B. Node and Link Embedding Vector Design

The input to the node/link embedding vector is a graphical representation of the problem $(\mathcal{G}, \mathbf{V}, \mathbf{E})$, i.e., the graph itself and node and link attributes. Matrix $\mathbf{V} \in \mathbb{R}^{|\mathcal{N}| \times d_{\text{v}}}$ stores the node-specific features, where the $i$-th row of the matrix is a $d_{\text{v}}$-dimensional vector representing the feature vector associated with node $i$. Likewise the link specific feature vectors are stored in matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}| \times d_{\text{e}}}$.

*1) Input Layer:* This step projects the node and link information to a higher dimension for later processing; it should include all the individual information that is beneficial for solving the problem. Each initial node embedding vector is derived from the 2-D geometric node coordinates. Then it is transformed into a $d_{\text{n-hid}}$-dimensional vector through a linear layer whose parameters are shared for all nodes. The initial embedding vector for node $i$ is

$$\mathbf{x}_i^0 = \text{LIN}_1(\text{pos}_i). \tag{8}$$

To obtain the initial embedding vector for a link, we define an indicator function $I(u, v)$ for link $(u, v)$. $I(u, v) = 1$ if either node u or v itself is or directly connected to a source or destination node of any commodity flow; $I(u, v) = 0$ otherwise. The $d_{\text{e-hid}}$-dimensional embedding vector of link $(u, v)$ is the concatenation of the end point's coordinates, the capacity and the indicator values after linear transforms:

$$\begin{aligned} \mathbf{y}_{(u,v)}^0 = \ & \text{LIN}_1(\text{pos}_u) \ || \ \text{LIN}_1(\text{pos}_v) \\ & || \ \text{LIN}_2(c(u, v)) \ || \ \text{LIN}_3(I(u, v)). \end{aligned} \tag{9}$$

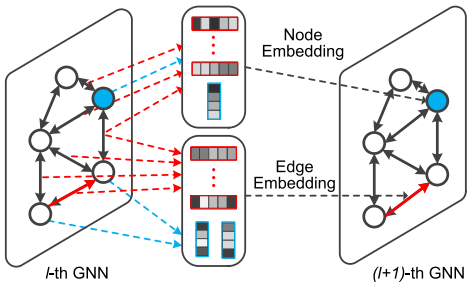
Each of the four parts have the same dimension $d_{\text{e-hid}}/4$. By doing this, we utilize the link capacity $c(u, v)$, which is essential for link usage; and also give an additional hint to the model $I(u, v)$ because links close to the source and destination nodes should be given more consideration than those that are not.

*2) Graph Convolution:* The node and link embedding vectors are updated iteratively for a fixed number of rounds $L$. In each iteration, the update is a weighted combination of the neighbors' embedding vectors, as shown in eqs. (10) and (11).

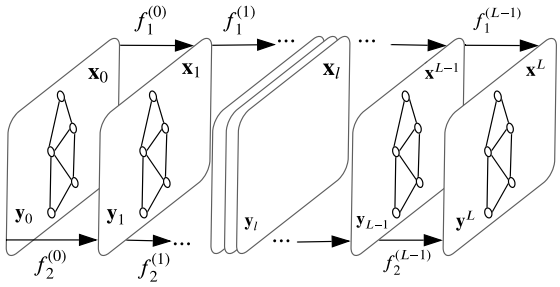
$$\mathbf{x}_u^{(l+1)} = f_1^{(l)}(\mathbf{x}_u^{(l)}, \frac{1}{|\mathcal{N}_u|} \sum_{v \in \mathcal{N}(u)} \mathbf{y}_{(u,v)}^{(l)}) \tag{10}$$

$$f_1^{(l)}(\mathbf{a}, \mathbf{b}) \triangleq \text{MLP}_1^{(l)}(\mathbf{a} + \text{MLP}_2^{(l)}(\mathbf{a} + \mathbf{b}\mathbf{W}_1^{(l)})) \tag{11}$$

In iteration $l$, the updated embedding vector $\mathbf{x}_u^{(l+1)}$ is a function $f_1^{(l)}$ of the current embedding vector and the average of the embedding vectors of the neighboring links. The function $f_1^{(l)}$ is the neural network that takes two input and processes them with two multi-layer perceptrons $\text{MLP}_1$ and $\text{MLP}_2$. The parameters include the matrix $\mathbf{W}_1 \in \mathbb{R}^{d_{\text{e-hid}} \times d_{\text{d-hid}}}$, and the internal parameters of $\text{MLP}_1$ and $\text{MLP}_2$. This form is chosen to be invariant with respect to the network size or index: the second argument $\mathbf{b}$ is the sum of the neighboring links, which remains constant with respect to how nodes and links are




(a) The proposed graph convolution layer operation in one iteration. The blue node's updated vector is shown to be a function of its own vector and its neighboring link's vectors; The red link's updated vector is a function of its own vector and its neighboring node's vectors.



(b) Using L iterations of GNN layer. Each board in the figure represents the node and link vectors at a given iteration. Iterations are connected by arrows representing the update functions, which are implemented by topology-aware neural networks.

Fig. 3. The proposed node and link embedding vector generation.

numbered. It is then combined with the node information $\mathbf{a}$ for the updated node vector.

The link embedding vectors are updated, and in the process are coupled with the node vectors. In the $(l+1)$-th iteration is shown in eqs. (12) to (15). The main neural network here is the function $f_2$ taking 4 inputs: the current link vector, the aggregation of the neighboring links, and the current endpoints' node vectors. Similarly, the function $f_2$ consists of two MLPs $\mathrm{MLP}_3$ and $\mathrm{MLP}_4$, with parameters $\mathbf{W}_2, \mathbf{W}_3 \in \mathbb{R}^{d_{\text{e-hid}} \times d_{\text{e-hid}}}$ and $\mathbf{W}_4, \mathbf{W}_5 \in \mathbb{R}^{d_{\text{n-hid}} \times d_{\text{n-hid}}}$. The input $\mathrm{AGG}(\mathcal{M}_{(u,v)})$ models the effects of other links to link $(u, v)$. It is a function operating on a set of neighboring links $\mathcal{M}_{(u,v)}$, which is the links with nodes that are within a distance $L_M$[2] of $u$ or $v$. The aggregation sums up the individual cross-link contribution, modeled as the element-wise product of $\mathbf{z}_{(u,v),(u\prime,v\prime)}$ and the current link vector $\mathbf{y}_{(u\prime,v\prime)}$:

$$\mathbf{y}_{(u,v)}^{(l+1)} = f_2^{(l)}(\mathbf{y}_{(u,v)}^{(l)}, \mathrm{AGG}(\mathcal{M}_{(u,v)}), \mathbf{x}_u^{(l)}, \mathbf{x}_v^{(l)}) \tag{12}$$

$$f_2^{(l)}(\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}) \triangleq \mathrm{MLP}_3^{(l)}(\mathbf{a} + \mathrm{MLP}_4^{(l)}((\mathbf{a}\mathbf{W}_2^{(l)} + \mathbf{b}\mathbf{W}_3^{(l)}) \| (\mathbf{c}\mathbf{W}_4^{(l)} + \mathbf{d}\mathbf{W}_5^{(l)})) \tag{13}$$

$$\mathrm{AGG}(\mathcal{M}_{(u,v)}) \triangleq \mathrm{MLP}_5^{(l)} \left( \sum_{(u\prime,v\prime)\in\mathcal{M}_{(u,v)}} \mathbf{z}_{(u,v),(u\prime,v\prime)} \odot \mathbf{y}_{(u\prime,v\prime)} \right) \tag{14}$$

$$\mathbf{z}_{(u,v),(u\prime,v\prime)} \triangleq \mathrm{MLP}_6^{(l)}((\mathbf{x}_u\mathbf{W}_6 + \mathbf{x}_{v\prime}\mathbf{W}_7) \| (\mathbf{x}_v\mathbf{W}_8 + \mathbf{x}_{u\prime}\mathbf{W}_9)). \tag{15}$$

And $\mathbf{z}$ acts as a mask that weighs how relevant link $(u\prime, v\prime)$ is to the current link $(u, v)$, and it is calculated by considering how one link's transmitter affects the other link's receiver, as shown in eq. (15). The update process is illustrated in fig. 3a.

Such node and link vector update is performed for a total of $L$ times, as shown in fig. 3b. These updates are implemented as $L$ neural networks with the same architecture but different parameters. The intuition behind repeating this operation for several rounds is that we hope to propagate topology information on different scales: in the first iteration, information about one-hop neighborhood is passed, and in the next a few iterations, for a given node or link, local information from faraway places is gradually refined into a high-level summary that aids in the decision making. Since the update processes information on different levels, we allow the training process to set these NNs to different parameters.

From eqs. (10) to (15), we can see that no matter how the nodes and links in the graph change their order or index in the input, the resulting link or node vectors would not be affected, because the computation is based on neighborhood exchange of information, and the neural network is not aware of any element order issue.

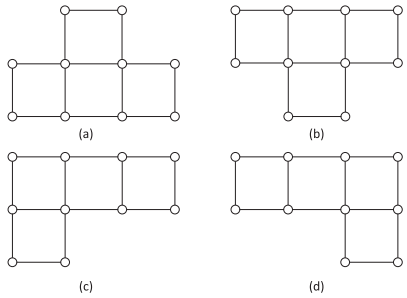
Note that our embedding technique in fact offers a flexible and generic method for embedding, and is significantly different from existing graph-convolutional embedding for wireless scheduling [20], [48]. The convolution filter in [48] explicitly assumes an interference pattern and focuses on embedding information within the immediate neighborhood only, while works like [20] assumes a fixed network scale in the convolution step. Our embedding methods incorporate node vector, link vector, and interactions among those vectors through parameters determined by training, and pose no restrictions on the network size to be processed. It is such a design that enables the embedding to code the impact of a certain link on the end-to-end (i.e. commodity flow level) performance under complex interference relationships, when the embedding is properly trained through solution samples from multi-hop wireless networks.

As an example, we give a visual demonstration of how the topology-aware embeddings can characterize the difference and similarity between network instances. We consider 4 types of network topologies, as shown in Figure 4a. For each topology, we generate 25 network instances by randomly perturbing the link capacities. For each type of the network instances, we apply the topology-aware graph embedding processing and project the embedding vectors to a 3-dimensional space.[3]

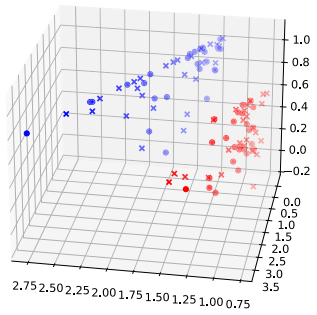
In Figure 4b, we plot the projected vectors. We use a combination of colors and marker styles to differentiate the four types of network instances: red dots, red 'x's, blue dots,

[2]Unless specified otherwise, $L_M$ is taken to be the interference range.

[3]We divide the embedding vector dimensions into three equal regions. The summation of the elements in the three region are then projected and plotted in the 3-dimensional space.




(a)



(b)

Fig. 4. Illustration of topology representation. (a): Four types of topology. (b): The efficiency of embedding vector in encoding and differentiating the settings of network topologies and link capacities. Red dots and red 'x' represent type (a) and type (b) networks, while blue dots and blue 'x' represent type (c) and type (d) networks.

and blue 'x's represent type (a), (b), (c), and (d) networks respectively.

We can see that type (a) and type (b) network instances cluster in the highly overlapped sub-space with similar distributions. Such structure-level similarity is expected, as topology (b) is just a vertical flip of topology (a). The structure similarity between type (c) and type (d) instances is also demonstrated, as topology (d) is a horizontal flip of topology (c). Furthermore, the clear separation between the blue sub-space and red-subspace indicates that our embedding successfully captures the distinction between two different topology structures.

This example illustrates that the graphical neural network has the capability to identify network instances that are similar in the structure. And we believe that this capability is key to solving the optimal scheduling problem, because with it, the neural network can associate the important scheduling links with the corresponding network topology. In other words, this capability enables the NN to learn the mapping from the input network information to the link importance scores.

In some sense, this example also justifies the superior performance observed in section V, by showing its ability to differentiate between different topologies. In section V, extensive numerical results will be presented to demonstrate the effectiveness of our embedding for the link prediction problem targeted in this paper, with comparison to the adjacency matrix based topology representation.

### C. Link Prediction With Attention

*1) Embedding Data Demands:* To encode the information of demand sets $\mathcal{C}$, we treat a source-destination node pair as if it is a virtual link. We use a feed-forward network with set-invariant properties [49] using the embedded nodes as the input, as shown in eq. (16)

$$\boldsymbol{q}_{\mathcal{C}} = \sum_{k \in \mathcal{C}} \mathrm{MLP}_q(\mathbf{x}_{s_k}^{(L)} \| \mathbf{x}_{t_k}^{(L)}), \tag{16}$$

where $\mathrm{MLP}_q$ is a learnable MLP block. For each demand node pair, the operation concatenates the final node embedding vectors of source and destination nodes and passes them through an MLP to obtain a $d_{\mathrm{d}}$-dim embedding vector. The final results are added as the vector representation of all the demands.

*2) Output Generation:* The prediction $\hat{\boldsymbol{y}} \in \mathbb{R}^{|\mathcal{E}|}$, with elements in $[0, 1]$ indicating the likelihood that the link is going to be relevant in the master problem, is generated with the attention mechanism. Specifically,

$$\hat{\boldsymbol{y}} = \mathrm{ReLU}(\mathrm{Attention}(\boldsymbol{q}_{\mathcal{C}} \boldsymbol{W}_q, \boldsymbol{Y}^{(L)} \boldsymbol{W}_v)) \tag{17}$$

with $\mathrm{Attention}(\mathbf{x}, \boldsymbol{Y}) \triangleq \mathbf{x}^T \mathbf{Y}$, where the demand set vector $\boldsymbol{q}_{\mathcal{C}}$ and the edge embedding are first converted to an equal dimension by the coefficients $\boldsymbol{W}_q$ and $\boldsymbol{W}_v$, and then use inner product followed by the non-linear ReLU activation function to get the final prediction.

Intuitively, this operation measures for each link how relevant they are under the current network topology and the given demand set, where the score is given by the inner product. In our implementation, links with a score higher or equal to a threshold $\alpha$ will be maintained; otherwise, the links are pruned from the topology.

Note that $\alpha$ is a hyper-parameter tuned for each dataset. Because it is not part of the neural network model, it can be considered as a post-processing parameter. By a binary grid search, we choose its value by finding one that maximizes the performance index, which will be defined in section V-A. The setting of $\alpha$ is listed in algorithm 1. There are valid $\alpha$ values in $(0, 1)$, and we want to find one best value to 2 decimal places. We do not try these 100 potential values, so instead we sample them, evaluate the model performance with these candidate values and only keep the best half of them to the next round. Iteratively, this leads to be a single best value. Although this is not theoretically guaranteed to always find the best value, it is a good trade-off between time cost and performance.

The stated neural network structure is different from works [50] in these important ways. First, the graph convolution is *link centric*. As our problem is focused on link-level decisions, and nodes do not provide much information other than the specify the flow endpoints, much of the information is around the links and their neighbors instead of nodes; second, our neural network structure is chosen on the nature of interference networks: we do not use a generic inner-product similarity measure in modeling the interactions of links, as done in machine learning works. Instead, we take note that the link interference relationship is 1) *range-limited*, so only links within a range are calculated; and 2) ultimately

**Algorithm 1** Setting $\alpha$ Value.

**Data**: Potential $\alpha$ values from $0.01, 0.02 \cdots 0.99$
**Result**: A single $\alpha$ value
Let $S_0 = \{0.01, 0.02, \cdots, 0.99\}$;
**for** $k = 0, 1, \cdots, \lceil \log |S_0| \rceil$ **do**
  Divide $S_k$ into 10 or fewer uniform subsets;
  Randomly select one point in each segment to evaluate the model performance;
  Select the best 5 segments to keep and discard the rest;
  Let $S_k$ be the union of the segments and sort them from lowest to highest;
**end**
Return the single value in $S_{\lceil \log |S_0| \rceil}$

determined by transmitter and receiver nodes' interactions, hence the form of eq. (15).

### *D. Customized Learning Techniques*

As the problem we study is of huge scale and of irregular form, we take these additional steps to ensure efficient training.

*1) Curriculum Training:* We take a *curriculum training* [51] approach to organize the samples and conduct training. The training samples obtained over different network sizes with a different number of commodity demands can be interpreted as have different levels of difficulty. A smaller network size or a smaller number of commodity demands implies a lower level of difficulty. Given a setting with a certain network size and number of commodity demands, we will generate training instances by properly arranging the locations of commodity source and destination node so that the training samples can cover a plenty variety of traffic patterns showing how traffic flow needs to be distributed within the network under different demand and interference scenarios. Furthermore, for cases with the same network size and the same number of commodity demands, the difficulty level is differentiated by the number of independent sets incurred in the optimization solution, which implicitly reflects the interference relationships in the case being evaluated.

With all these efforts, we then utilize samples of different difficulty for different training epochs. The difficulty increases as the training goes further. Specifically, before the training, we sort the samples in the training dataset according to how much time the conventional algorithm takes to solve the sample instance, from the lowest to highest. The samples are divided into $N_{\text{parts}} = 4$ equal parts, representing four subdivisions of data with increasing difficulty. At first, the model is trained only with data from the easiest first part. Every time a given number of epochs is passed, the training data to be used is joined by the next, more difficult part. In the last part of the training, the entire training dataset is used, just like the ordinary training procedure without curriculum training.

*2) Loss Function:* Since the ultimate test of learning performance, which is the optimum solution to the problem (6) given the current estimate of link usefulness, is a non-differentiable function of the model parameters, we instead use a differentiable proxy measure, a modified cross-entropy. It is the sum of binary cross-entropy of the individual link's usage distribution as

$$\mathcal{L} = \sum_{i=1}^{|\mathcal{E}|} w_i \big( y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \big), \qquad (18)$$

where $y$ and $\hat{y}$ are respectively the actual and predicted link values; $w_i$ is the link-wise weight coefficient. This value is calculated on batch sample basis when using SGD (stochastic gradient descent) algorithm to minimize.

Note that the purpose of (18) is essentially to facilitate performing a multi-label classification on the links: given a network topology $G$ and a demand set $\mathcal{C}$, for each of the link $l$, the goal is to use neural networks to approximate the conditional probability $\hat{y}_l = p_l(\text{use} = 1 | G, \mathcal{C})$. The likelihood of observing the link usage pattern is therefore $\prod_l \hat{y}_l^{y_l} (1 - \hat{y}_l)^{1 - y_l}$. Equivalently, its negative logarithm is the exact form of cross-entropy.

The motivation of incorporating the link-wise weight coefficient is to mitigate the prediction errors over those highly important links. Specifically, we assign the weight coefficient over link $l$ as $w(l) = \beta \times \text{normalized flow} = \frac{\beta \sum_{k=1}^{K} f_k(l)}{\sum_{k=1}^{K} r_k}$.

It means the link weight coefficient for link $l$ is proportional to the ratio between the total network flow carried over link $l$ and the total commodity throughput over the network. The parameter $\beta$ can be considered as a parameter that can tune the sensitivity of the weight coefficient in affecting the learning performance. $\beta$ can be adjusted in practice; in our experiment, we find setting $\beta = 10$ is a good choice.

*3) Feasibility Guarantee:* Note that the trained machine predicts important links in the probabilistic sense. There is a possibility that inaccurately pruned links might hurt the network connectivity and thus impact the feasibility of the network optimization. In our implementation, given a topology, we pre-compute a path for each commodity flow. The subset of links finally selected (which will be fed to the optimization solver) is the union of the subset of links survived from machine pruning and the links from all pre-computed commodity shortest paths. With such an approach, the union of pre-computed shortest paths ensures network connectivity, while the incorporation of important links predicted by a machine ensures the quality of optimization. We would like to emphasize that the above operations for feasibility guarantee will not cause any extra cost to the training procedure. The pre-computation of commodity paths can be interpreted as a pre-processing step when the trained machine is applied over a new topology. If there are a large number of application scenarios over the same topology to be evaluated, this pre-processing is just a one-time cost.

## V. Numerical Experiments

### *A. Experiment Setup*

To obtain training data in a topology-aware context, we compute a large number of optimization instances over various network topologies and commodity flow deployments

TABLE II
THE LIST OF PARAMETERS USED IN INSTANCE GENERATION AND TRAINING

| Name | Value |
|---|---|
| Transmission Range | 70 m |
| Interference Range | 50 m |
| Minimum Distance | 20 m |
| Area Side Length | 1000,2000 m |
| Transmission Power | 1mW |
| Maximum Connections per Node | 10 |
| Number of Nodes | 10,30,50,200 |
| Avg. Number of Links | 26, 102, 325, 2420 |
| Number of Flow Commodities | 1-5 |
| Epochs | 200 |
| Learning rate | 0.0002 |
| Batch Size | 16 |
| L | 4 |
| $d_{n-hid}$ | 96 |
| $d_{e-hid}$ | 192 |
| $\text{MLP}_1$, $\text{MLP}_2$ dimensions | $96 \times 96 \times 96$ |
| $\text{MLP}_3$, $\text{MLP}_4$ dimensions | $192 \times 192 \times 192$ |
| $\text{MLP}_5$ dimensions | $192 \times 192 \times 192$ |
| $\text{MLP}_6$ dimensions | $192 \times 192$ |
| $\text{MLP}_q$ dimensions | $192 \times 256 \times 300$ |
| $\mathbf{W}_1$ | $192 \times 96$ |
| $\mathbf{W}_2$, $\mathbf{W}_3$ | $192 \times 128$ |
| $\mathbf{W}_4$, $\mathbf{W}_5$ | $96 \times 64$ |
| $\mathbf{W}_6$, $\mathbf{W}_7$ | $96 \times 96$ |
| $\mathbf{W}_8$, $\mathbf{W}_9$ | $96 \times 96$ |
| $\mathbf{W}_q$, $\mathbf{W}_v$ | $70 \times 300, 192 \times 300$ |

by the DCG algorithm [41]. All these instance solutions will be mixed together to form a training data set to conduct supervised learning. The trained machine will then be applied for link prediction over different new topologies and commodity flow cases to examine the topology-aware efficiency and robustness of our learning methods.

*1) Datasets:* The network instances are generated following different rules: in the `random` dataset, the network topology forms a *random geometric* graph: a given number of nodes are uniformly randomly placed within a square area, with minimum and maximum distances; in the `grid` dataset, the nodes are placed on a rectangular grid with a Gaussian random perturbation to account for the imperfections of the real-world deviations. The nodes are considered *connected* to as many neighbors as possible, up to a given number representing the maximum connections per node, within their transmission range. And any of these nodes can be chosen as the next hop by the scheduling process. For each case, the nodes with data demands are selected randomly, and the total number of demands ranges from $1-5$. We generate data samples with 10, 30, 50, and 200 nodes to represent the cases small, medium, and large networks.

Although the model can be trained from scratch with purely random network instances, we choose to differentiate between different types of topology in data preparation. This is because certain types, e.g., the grid networks, are commonly encountered in network applications but not well represented by a random network generator. However, in the training process, there is no distinction in how the model calculates the decisions based on the instance generation rules. We list the parameters used for instance generation in the Table II.

*2) Comparisons:* We experiment with three additional methods to compare with our TADL method:

- `SAGECONV`, the embedding units in TADL are replaced with the node-based graph invariant embedding model developed in [50]; The demand information is included as node labels. It represents the scheme where the embedding vectors only consider node neighborhood information without using the demand as the global information and making link-based decisions. It does not encode commodity-level information, while our embedding technique as presented in Section 3 incorporates commodity-level information.
- `ADJ`, the learning framework in [8] is enhanced with adjacency matrix as topology information;
- `Blind`, a topology-blind scheme where some links are arbitrarily activated (without machine learning), with a fixed probability that is equal to the frequency of the activated links in the training.

For all experiments, these methods' performance metrics are calculated from the same training, evaluation, and testing datasets. The same technique stated in section IV-D3 to ensure the feasibility is used for all the methods. We examine the performance mainly by two metrics. Essentially, we aim to quantify how much speed up the neural model can achieve by cutting out irrelevant network links and causing how large a loss in optimality, measured as a fraction of the optimal value obtained from the conventional method. These metrics are defined as follows.

*a) Approximation Ratio (AR):* it is defined as the ratio between the optimal network flow computed on the reduced problem instance, denoted as $\text{OPT}_{\text{pruned}}$, and that computed over the original problem, denoted as $\text{OPT}_{\text{original}}$: $r_{\text{approx}} = \text{OPT}_{\text{pruned}}/\text{OPT}_{\text{original}}$. It shows how the pruned problem instances approach the optimal network capacity.

*b) Computation Time Reduction (TR):* given a problem instance, we use $t_{\text{org}}$ to denote the computation time solving the original problem. When a pruning method is implemented, the effective computation time will be $t_{\text{ML}} = t_{\text{setup}} + t_{\text{inference}} + t_{\text{reduced instance}}$. In this definition, $t_{\text{setup}}$ is the time for setting up the model and other bookkeeping tasks. In our case, it is mainly the time for computing the connected paths in the network to ensure solution feasibility; $t_{\text{inference}}$ is the time for the trained model to prune links (which is 0 for the `Blind` algorithm); $t_{\text{reduced instance}}$ is the time for solving the reduced-sized instance. The computation time reduction radio is defined as $r_{\text{red}} = (t_{\text{org}} - t_{\text{ML}})/t_{\text{org}}$.

*c) Performance Index (PI):* We use the following formula as a composite measure of both AR and TR:

$$\text{PI} = \frac{\log(1 + r_{\text{approx}}) + \log(1 + 0.6\ r_{\text{red}})}{\log(2) + \log(1.6)}. \tag{19}$$

This measure has these convenient properties that are easily derived from the form: 1) PI is valued between zero and one; 2) PI is monotonously increasing in AR and TR, so improving either figure contributes to overall performance; 3) PI has a diminishing return for either AR and TR, encouraging the system to make a better trade-off between the two measures; 4) PI is biased by the coefficient 0.6 to slightly favor having

TABLE III

APPROXIMATION RATIO AND TIME REDUCTION RATIO COMPARISON FOR RANDOMLY GENERATED CASES WITH DIFFERENT INSTANCE SIZES. BOTH PERFORMANCE METRICS ARE THE LARGER THE BETTER. FOR EACH NETWORK SIZE, WE LIST THE SOLUTION TIMES, IN SECONDS. IN EACH ENTRY, THE FIRST NUMBER IS THE SOLUTION TIME USING TADL AND THE SECOND NUMBER IS THE CONVENTIONAL ALGORITHM'S SOLUTION TIME

| | 10 | | | 30 | | | 50 | | | 200 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 3 | 5 | 1 | 3 | 5 | 1 | 3 | 5 | 1 | 3 | 5 |
| TADL-AR | 0.96 | 0.94 | 0.93 | 0.95 | 0.91 | 0.92 | 0.92 | 0.91 | 0.88 | 0.87 | 0.86 | 0.75 |
| TADL-TR | 0.78 | 0.72 | 0.69 | 0.67 | 0.64 | 0.58 | 0.66 | 0.62 | 0.60 | 0.76 | 0.72 | 0.74 |
| SAGECONV-AR | 0.82 | 0.87 | 0.81 | 0.84 | 0.77 | 0.72 | 0.74 | 0.73 | 0.71 | 0.84 | 0.82 | 0.81 |
| SAGECONV-TR | 0.74 | 0.69 | 0.52 | 0.71 | 0.61 | 0.54 | 0.58 | 0.54 | 0.50 | 0.68 | 0.52 | 0.66 |
| BLIND-AR | 0.67 | 0.72 | 0.72 | 0.68 | 0.64 | 0.66 | 0.56 | 0.51 | 0.43 | 0.71 | 0.74 | 0.61 |
| BLIND-TR | 0.42 | 0.40 | 0.37 | 0.48 | 0.35 | 0.35 | 0.37 | 0.33 | 0.29 | 0.39 | 0.42 | 0.38 |
| ADJ-AR | 0.78 | 0.71 | 0.7 | 0.71 | 0.67 | 0.62 | 0.72 | 0.73 | 0.71 | 0.75 | 0.74 | 0.65 |
| ADJ-TR | 0.38 | 0.32 | 0.18 | 0.23 | 0.11 | 0.12 | 0.22 | 0.27 | 0.20 | 0.25 | 0.32 | 0.31 |
| Average Solution Time | | 0.08/0.30 | | | 0.23/0.69 | | | 1.22/3.43 | | | 9.27/31.84 | |
| TADL-PI | 0.91 | 0.88 | 0.86 | 0.86 | 0.83 | 0.81 | 0.84 | 0.82 | 0.80 | 0.86 | 0.84 | 0.81 |
| SAGECONV-PI | 0.83 | 0.83 | 0.74 | 0.83 | 0.76 | 0.71 | 0.73 | 0.71 | 0.68 | 0.82 | 0.74 | 0.80 |
| BLIND-PI | 0.63 | 0.64 | 0.63 | 0.71 | 0.63 | 0.60 | 0.63 | 0.60 | 0.52 | 0.64 | 0.66 | 0.58 |
| ADJ-PI | 0.66 | 0.60 | 0.53 | 0.56 | 0.48 | 0.46 | 0.56 | 0.59 | 0.56 | 0.59 | 0.63 | 0.58 |

a high approximation ratio because we consider the solution quality to be more important than additional time cost reduction.

### *B. Computation Time Reduction and Approximation Ratio Trade-off*

To give a general idea of how the model performs, we group the instances into different groups according to their sizes, and then train and evaluate the models separately. Within each group, instances have different flow demands, node and link numbers to observe the performance under different data configurations. The testing is performed with the samples not seen by the training process from the same dataset.

In Table III, we demonstrate average values of approximation ratios and time reduction in testing dataset that is not seen by the training process. The table is grouped according to the number of the network nodes and then subdivided by the number of commodity flows. Each network size corresponds to a trained model, and we report the performance figures for each number of flows separately.

With our method TADL, we can observe a significant solution time reduction, over sixty percent, while maintaining a high solution quality at a minimum of 75%, across a wide range of network sizes, up to 200 nodes (with an average of 2420 links). Compared with other methods, TADL method robustly achieves a better trade-off between optimization quality and complexity reduction, as demonstrated by the high PI across various network sizes and demands. We also notice that that because the threshold value is tuned for good PI and because PI is biased to favor good AR, all the methods have a reasonable level of AR. However, the time reduction for TADL is markedly higher than the other methods, suggesting that it is better at identifying relevant links that are truly useful without adding in other useless links.

Methods with embedding (i.e., TADL and SAGECONV) technique perform better than the methods without, but our method of combining link and node embedding achieves an overall better performance. The BLIND method results in a low time reduction at around 40% due to the fact lacking the understanding of the network topology, it cuts off many necessary links, which would have to be added in the feasibility processing step; the net effect is that the network size is not reduced by a large proportion. In almost all the scenarios, the adjacency matrix method `ADJ` performs similar or even worse than `Blind`, in terms of PI. The results confirm our analysis before that directly using an adjacency matrix as the neural network input cannot give an accurate indication of the topology change and thus hard to extract meaningful topology related information to facilitate link prediction.

From the results, there is a general trend that the approximation ratio is lowered as the number of nodes rises. This is expected given that the difficulty of the problem is higher and it is more likely to miss a high quality solution. The time reduction shows a similar trend, because less important links may wrongly remain to add to the time cost. We note that in the domain of high node count (50 and above), the TR is still quite high (at least 60%), suggesting that removing unnecessary links results removes a bottleneck in the time performance.

The impact of the number of traffic demands on the performance is similar: generally, lower traffic demand cases perform better than cases with more traffic demands. This is because, with a higher number of flow demands, the overall scheduling is more fragmented, as at any time only a link can only carry one flow. This causes the scheduling task to take more iterations to discover a good set of ISs, and that there are more links that need to be considered in the solution. Accordingly, the time reduction and approximation ratio in low flow count cases are the most significant because the model can accurately infer the needed links.

### *C. Performance Robustness to Network Scale*

To see how the model performs when we use the model trained from one scale to another, we apply the model trained on 200-node cases and test them on smaller instances without re-training, and compare them with the performance from the models trained on cases with their respective sizes. This is to

TABLE IV

APPROXIMATION RATIO AND TIME REDUCTION RATIO OF CROSS-NODE SIZE PERFORMANCE. THE COLUMNS "ORIGINAL" LISTS PERFORMANCE FIGURES OF THE MODEL TRAINED SEPARATELY FOR EACH NODE SIZE; THE COLUMNS "CROSS" COMES FROM THE MODEL TRAINED ON 200-NODE CASES

| | 10 | | 30 | | 50 | |
|---|---|---|---|---|---|---|
| | cross | original | cross | original | cross | original |
| AR | 0.91 | 0.94 | 0.86 | 0.93 | 0.84 | 0.90 |
| TR | 0.70 | 0.73 | 0.59 | 0.63 | 0.60 | 0.63 |
| PI | 0.81 | 0.84 | 0.73 | 0.78 | 0.72 | 0.77 |

see if the model is able to extract useful knowledge of the problem such that with different sizes the benefit still persists.

We can observe from Table IV that the cross-node size performance is very close to the separately trained models. This implies that by training from sufficiently diverse data, the network scale does not inherently limit the performance of this scheme. This discovery makes it possible to train from large and difficult instances and the model can adapt to other easier cases with no need for retraining.

Moreover, the cross-network size performance is correlated with the difficulty of the cases: in the small networks where solution comes easier, the performance is better than the more difficult cases that correspond to larger networks.

### D. A Practical Office Setting

To further examine the robustness of TADL, we apply it over a wireless mesh topology with 2 commodity flows, in a practical office setting which was studied in [28] as shown in Figure 5. In the topology, each edge represents a bi-directional link, thus giving 96 unidirectional links in total.

This result is obtained directly from the model that was trained on the dataset with 30-node instances, and the office setting is not part of the dataset. TADL generates a reduced problem of 29 links with the approximation ratio $r_{\text{approx}} = 0.97$ and the time reduction ratio $r_{\text{red}} = 0.76$.

Figure 5 also indicates the exact set of links that are activated in the optimal solution from the original problem to benchmark the prediction accuracy. We can tell that TADL only incorrectly prunes a few links and include a small set of redundant links. Note that after link pruning, a sparse problem instance such as this example can become disconnected due to the probabilistic nature of the solution; our proposed method can handle this situation by feasibility processing.

### E. Energy Efficiency Benefits

Although multi-commodity flow is a classic topic that interests network researchers, nowadays the network's energy efficiency receives even more attention. In this section, we examine how the TADL method is of practical benefit in this aspect.

We define the energy efficiency of the network to be the ratio of total transmitted bits to the energy consumption of the network nodes and links, within a unit time. The power consumption of network is modeled as the sum of dynamic and static parts. For the static part, each node is assumed to have a 0.05 mW consumption regardless of its operating

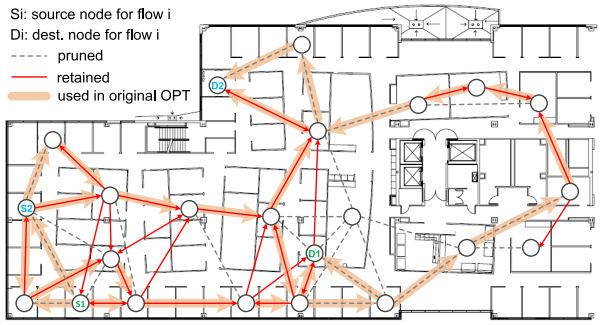



Fig. 5. A wireless mesh topology in an office setting constructed in [28]. TADL link prediction and the optimal link set are shown.

TABLE V

ENERGY EFFICIENCY COMPARISON FOR CASES WITH RANDOM TOPOLOGY AND A REAL-WORLD DATASET TOPOLOGY, WITH DIFFERENT AMOUNT OF FLOWS. THE NUMBERS REPRESENTING THE ENERGY EFFICIENCY ARE IN MBITS/JOULE AND ARE THE LARGER THE BETTER

| | 50-rand | | 200-rand | | RTD |
|---|---|---|---|---|---|
| Algo/No.flows | 3 | 5 | 3 | 5 | 5 |
| TADL-EE | 4.53 | 4.04 | 1.55 | 1.45 | 6.10 |
| SAGECONV-EE | 4.26 | 3.63 | 1.50 | 1.40 | 5.90 |
| BLIND-EE | 2.72 | 2.12 | 1.35 | 1.34 | 5.25 |
| ADJ-EE | 3.10 | 3.50 | 1.32 | 1.38 | 5.39 |
| Base | 3.73 | 3.45 | 1.35 | 1.27 | 5.32 |

condition. This is to model the overhead of each node's control circuit and application-specific functionality. The dynamic part accounts for the activation and deactivation of transmission links, as a result of the scheduling and routing decisions. For any given link $(u, v)$ where $u$ is the transmitter node and $v$ is the receiver node, $P_{dyn}(u)$ is set to the default transmission power (this parameter is also used in calculating link capacity), and $P_{dyn}(v)$ is set to a fixed amount of 5 mW.[4]

For better illustration, we introduce another external dataset featured in a multi-hop network deployment study [53], hereafter referred to as RTD. We make use of the provided network layout and node positions, which are based on the placement of wireless transmitters indoors. The source and destination nodes are randomly assigned, and the model is directly used from one trained on 30-node instances.

We calculate the energy efficiency achieved on different types of instances and report the findings in table V. We experimented in randomly generated 50- and 200-node instances, and also the cases from RTD [53]. From the table one can see that TADL has a clear advantage in terms of the energy efficiency over other methods. The row marked as "base" referred to using the original DCG algorithm. This is due to the fact it significantly cut down the number of active links while maintaining a reasonable throughput.

### F. Training Convergence

The training loss and the prediction quality of the proposed method and a closely related scheme based on graph learning, SAGECONV are illustrated in Figure 6.

[4]All power consumption values estimated from the manufacturer's data sheets of typical wireless sensor modules, e.g. LTP5901-IPM [52].

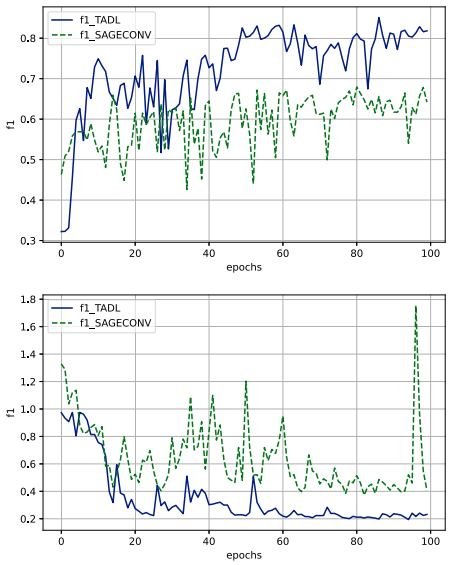



Fig. 6. The loss and f1 metric during training.

We use *F1* score[5] as the figure to represent the model's capability to balance between false positive and false negative. The data was taken when training the model on 50-node datasets with a mixed number of demands, a scale that is not trivial for the model to solve and also commonly encountered in the application.

We can observe that for our method, the descent of loss and the ascent of prediction F1 score happens at a higher slope and generally a few epochs earlier than SAGECONV. This observation indicates that our proposed method combining the local and global network information is more suited with this type of problem.

### G. Effects of Loss Function Choice and Curriculum Training

In addition, we study whether using a different loss function and a curriculum training scheme is beneficial for our problem. For each network size, we compare the performance index of models trained with a *sample weighted* loss function and those trained with an ordinary cross-entropy loss function. It is clearly observed that the former is a better choice from Figure 7. Fixing all the parameter settings except the loss function used, there is an observable performance gap between the weighted and unweighted loss function. This can be explained by the fact that the class imbalance in the training data is serious enough that additional weighting on the links to be used can help contribute to the final metric which the user cares the most.

Next, we examine the performance difference to see the effects of using curriculum learning. In a similar setting, we train models that use a curriculum training scheme based on sample weights, and also models that randomly select samples in each batch with no consideration of its difficulty. We plot the different performance indexes for several network scales and traffic flows in fig. 8. There is a slight increase in the overall performance favoring the usage of curriculum training, especially when the number of traffic s is high. In the plot we use a separate bar for the cases with 5 traffic flows to stress the performance difference.

[5]The F1 score is the harmonic mean of precision and recall: $F_1 = \frac{2}{\text{recall}^{-1}+\text{precision}^{-1}}$. Following the typical usage, we consider the *precision* "the number of links used by the optimal solution and correctly predicted by the model / number of links predicted important by the model," and *recall* as "the number of links correctly predicted by the model / the number of links used by the optimal solution." A number between 0 and 1, a larger F1 score indicates that the model identifies more important links without blindly predict all links to be useful.

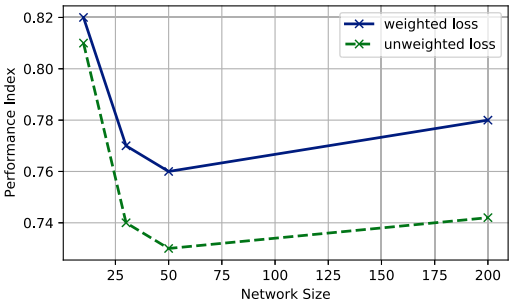



Fig. 7. Effects of using a weighted loss function versus without using such a loss function. By using sample weights, there is an average 4% improvement in the performance index.

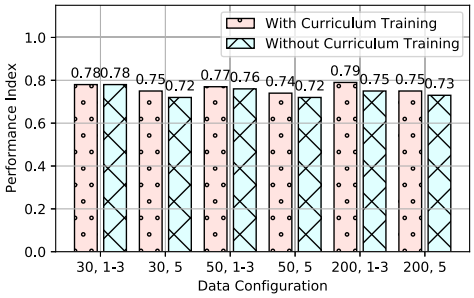
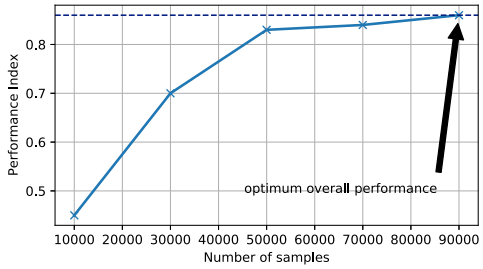



Fig. 8. Effects of using curriculum training. In each label, the first part shows the amount of network nodes, and the second part shows the amount of traffic flow demands. The cases with 1 and 3 flows are grouped together while 5 flow cases are separately plotted.

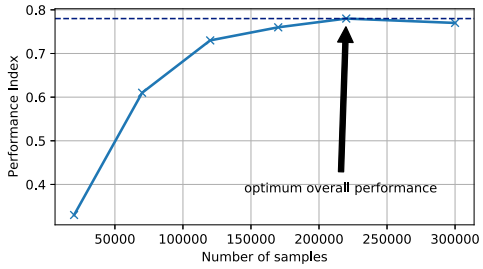
### H. Model Performance Versus Sample Volume

Since learning-based prediction is inherently a data-oriented approach, the model performance generally improves as the number of training samples increases. To verify this, we train models under different network sizes, with the same learning setting but different numbers of sample volumes in the training dataset. For each network size, the performance of the models is tested on the same set of non-training cases' data.

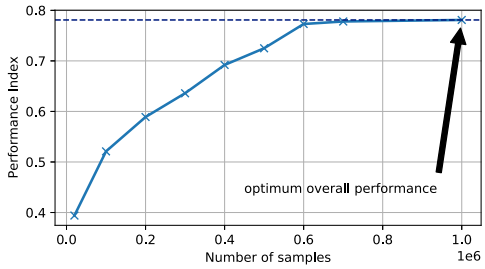
In Figure 9, we plot the performance index (PI) versus the training dataset size because it is a composite metric of the model quality. The results confirm that the performance index improves with the number of training data, but saturates at levels that decrease as the problem sizes increase. This observation shows that the model's capacity is not fully utilized when the number of training samples is not sufficient. Before a reaching the plateau, more data leads to significant performance improvement than tweaks in the model architecture. Beyond that, the model performance benefits little from additional training data. This can be attributed to the model becoming unable to improve its estimation of the current data's input-output correlation. More data beyond this point is not




(a) 10 node networks




(b) 50-node networks




(c) 200-node networks

Fig. 9. Model performance with changing number of training samples. The performance index, a number between 0 and 1, represents both the time reduction and approximation ratio. The optimum performance and the corresponding sample size are annotated by the arrow.

useful unless the model's architecture or parameter settings are changed.

## VI. Conclusion

This work contributes a topology-aware approach to train a DL machine which can robustly predict important links (and thus prune non-critical links) to facilitate wireless network optimization over dynamic network topologies. Efficient embedding techniques are developed to address the fundamental issue of index-independent topology representation. Our method can work in a complementary manner with the traditional theme of approximation algorithms, to further reduce computation complexity from a new dimension by leveraging historical computation data. As a next step, we will conduct a further in-depth topology-aware study in the more generic setting of multi-radio multi-channel wireless networks.

## References

[1] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu, "Impact of interference on multi-hop wireless network performance," *Wireless Netw.*, vol. 11, no. 4, pp. 471–487, Jul. 2005.

[2] K. N. Ramachandran, E. M. Belding, K. C. Almeroth, and M. M. Buddhikot, "Interference-aware channel assignment in multi-radio wireless mesh networks," in *Proc. INFOCOM*, vol. 6, Apr. 2006, pp. 1–12.

[3] L. Badia, A. Erta, L. Lenzini, and M. Zorzi, "A general interference-aware framework for joint routing and link scheduling in wireless mesh networks," *IEEE Netw.*, vol. 22, no. 1, pp. 32–38, 2008.

[4] O. Goussevskaia, Y.-A. Pignolet, and R. Wattenhofer, "Efficiency of wireless networks: Approximation algorithms for the physical interference model," *Found. Trends Netw.*, vol. 4, pp. 313–420, Mar. 2010.

[5] D. Chafekar, V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, "Approximation algorithms for computing capacity of wireless networks with SINR constraints," in *Proc. 27th Conf. Comput. Commun. (INFOCOM)*, Apr. 2008, pp. 1166–1174.

[6] S. Misra, S. D. Hong, G. Xue, and J. Tang, "Constrained relay node placement in wireless sensor networks: Formulation and approximations," *IEEE/ACM Trans. Netw.*, vol. 18, no. 2, pp. 434–447, Apr. 2010.

[7] R. Gandhi, Y. A. Kim, S. Lee, J. Ryu, and P. J. Wan, "Approximation algorithms for data broadcast in wireless networks," *IEEE Trans. Mobile Comput.*, vol. 11, no. 7, pp. 1237–1248, Jul. 2012.

[8] L. Liu, B. Yin, S. Zhang, X. Cao, and Y. Cheng, "Deep learning meets wireless network optimization: Identify critical links," *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 1, pp. 167–180, Mar. 2020.

[9] Q. Ye, W. Shi, K. Qu, H. He, W. Zhuang, and X. Shen, "Joint RAN slicing and computation offloading for autonomous vehicular networks: A learning-assisted hierarchical approach," *IEEE Open J. Veh. Technol.*, vol. 2, pp. 272–288, 2021.

[10] Q. Ye, W. Shi, K. Qu, H. He, W. Zhuang, and X. S. Shen, "Learning-based computing task offloading for autonomous driving: A load balancing perspective," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2021, pp. 1–6.

[11] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access in wireless networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 4, no. 2, pp. 257–265, Jun. 2018.

[12] Y. Yu, T. Wang, and S. C. Liew, "Deep-reinforcement learning multiple access for heterogeneous wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1277–1290, Jun. 2017.

[13] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, and P. Li, "Energy-efficient scheduling for real-time systems based on deep Q-learning model," *IEEE Trans. Sustain. Comput.*, vol. 4, no. 1, pp. 132–141, Mar. 2017.

[14] S. Chinchali *et al.*, "Cellular network traffic scheduling with deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 1–9.

[15] Z. Xu *et al.*, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. IEEE INFOCOM*, Apr. 2018, pp. 1871–1879.

[16] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proc. ACM SIGCOMM*, 2018, pp. 191–205.

[17] M. A. Wijaya, K. Fukawa, and H. Suzuki, "Neural network based transmit power control and interference cancellation for mimo small cell networks," *IEICE Trans. Commun.*, vol. 99, no. 5, pp. 1157–1169, 2016.

[18] F. Tang *et al.*, "On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control," *IEEE Wireless Commun.*, vol. 25, no. 1, pp. 154–160, Feb. 2017.

[19] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for interference management," *IEEE Trans. Signal Process.*, vol. 66, no. 20, pp. 5438–5453, Oct. 2018.

[20] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, "A graph neural network approach for scalable wireless power control," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2019, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/9024538/

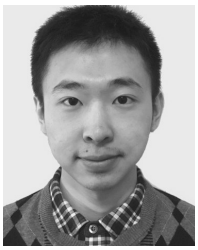
[21] C. Joo, X. Lin, and N. B. Shroff, "Understanding the capacity region of the greedy maximal scheduling algorithm in multi-hop wireless networks," in *Proc. 27th Conf. Comput. Commun. (INFOCOM)*, Apr. 2008, pp. 1103–1111.

[22] S. Kwon and N. B. Shroff, "Analysis of shortest path routing for large multi-hop wireless networks," *IEEE/ACM Trans. Netw.*, vol. 17, no. 3, pp. 857–869, Jun. 2009.

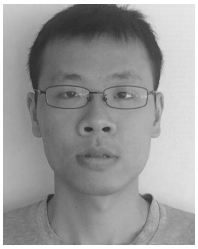
[23] L. Liu, Y. Cheng, L. Cai, S. Zhou, and Z. Niu, "Deep learning based optimization in wireless network," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2017, pp. 1–6.

[24] L. Babai, "Graph isomorphism in quasipolynomial time," in *Proc. ACM STOC*, 2016, pp. 684–697.

[25] Z. Han and K. R. Liu, *Resource Allocation for Wireless Networks: Basics, Techniques, and Applications*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[26] L. Georgiadis, M. J. Neely, and L. Tassiulas, "Resource allocation and cross-layer control in wireless networks," *Found. Trends Netw.*, vol. 1, no. 1, pp. 1–144, 2006.
[27] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, "Algorithmic aspects of capacity in wireless networks," in *Proc. ACM SIGMETRICS*, 2005, pp. 133–144.
[28] R. Draves, J. Padhye, and B. Zill, "Routing in multi-radio, multi-hop wireless mesh networks," in *Proc. ACM MobiCom*, 2004, pp. 114–128.
[29] L. Liu, Y. Cheng, X. Cao, S. Zhou, Z. Niu, and P. Wang, "Joint optimization of scheduling and power control in wireless networks: Multi-dimensional modeling and decomposition," *IEEE Trans. Mobile Comput.*, vol. 18, no. 7, pp. 1585–1600, Aug. 2018.
[30] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Belmont, NV, USA: Athena Scientific, 1998.
[31] J. Su, S. He, and Y. Wu, "Features selection and prediction for IoT attacks," *High-Confidence Comput.*, vol. 2, no. 2, Jun. 2022, Art. no. 100047.
[32] Q. Xia, W. Ye, Z. Tao, J. Wu, and Q. Li, "A survey of federated learning for edge computing: Research problems and solutions," *High-Confidence Comput.*, vol. 1, no. 1, Jun. 2021, Art. no. 100008.
[33] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 50–56.
[34] J. Jiang, S. Sun, V. Sekar, and H. Zhang, "Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation," in *Proc. USENIX NSDI*, 2017, pp. 393–406.
[35] B. Matthiesen, A. Zappone, K.-L. Besser, E. A. Jorswieck, and M. Debbah, "A globally optimal energy-efficient power control framework and its efficient implementation in wireless interference networks," 2018, *arXiv:1812.06920*.
[36] M. Lee, G. Yu, and G. Ye Li, "Graph embedding based wireless link scheduling with few training samples," 2019, *arXiv:1906.02871*.
[37] H. Li, Y. Cheng, C. Zhou, and P. Wan, "Multi-dimensional conflict graph based computing for optimal capacity in MR-MC wireless networks," in *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, Jun. 2010, pp. 774–783.
[38] Y. Cheng, H. Li, D. M. Shila, and X. Cao, "A systematic study of maximal scheduling algorithms in multiradio multichannel wireless networks," *IEEE/ACM Trans. Netw.*, vol. 23, no. 4, pp. 1342–1355, Aug. 2015. [Online]. Available: http://ieeexplore.ieee.org/document/6824270/
[39] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Trans. Inf. Theory*, vol. 46, no. 2, pp. 388–404, Mar. 2000.
[40] D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*, vol. 6. Belmont, MA, USA: Athena Scientific, 1997.
[41] Y. Cheng, X. Cao, X. Shen, D. M. Shila, and H. Li, "A systematic study of the delayed column generation method for optimizing wireless networks," in *Proc. 15th ACM Int. Symp. Mobile ad hoc Netw. Comput. (MobiHoc)*, 2014, pp. 23–32.
[42] T. Hastie, R. Tibshirani, and J. Friedman, *Elements of Statistical Learning*, 2nd ed. New York, NY, USA: Springer, 2001.
[43] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.
[44] Q. Zhang, R. Cao, F. Shi, Y. Nian Wu, and S.-C. Zhu, "Interpreting CNN knowledge via an explanatory graph," 2017, *arXiv:1708.01785*.
[45] U. Schöning, "Graph isomorphism is in the low hierarchy," *J. Comput. Syst. Sci.*, vol. 37, no. 3, pp. 312–323, Dec. 1988.
[46] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," 2017, *arXiv:1704.01212*.
[47] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
[48] W. Cui, K. Shen, and W. Yu, "Spatial deep learning for wireless scheduling," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1248–1261, Jun. 2019.
[49] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep Sets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3391–3401.
[50] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. ACM SIGKDD*, 2018, pp. 974–983.
[51] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. ACM ICML*, 2009, pp. 41–48.
[52] *SmartMesh IP Wireless 802.15.4e PCBA Module With Antenna Connector*, Analog Devices, Norwood, MA, USA, 2013.
[53] M. R. Souryal, J. Geissbuehler, L. E. Miller, and N. Moayeri, "Real-time deployment of multihop relays for range extension," in *Proc. 5th Int. Conf. Mobile Syst., Appl. Services*, 2007, pp. 85–98.



**Shuai Zhang** (Student Member, IEEE) received the B.Eng. degree from Zhejiang University in 2013, the M.S. degree from the University of California at Los Angeles, Los Angeles, in 2015, and the Ph.D. degree in computer engineering from the Illinois Institute of Technology in 2022. His research interests include wireless communication and distributed learning.



**Bo Yin** (Student Member, IEEE) received the B.E. degree in electronic information engineering and the M.E. degree in electronic science and technology from Beihang University in 2010 and 2013, respectively, and the Ph.D. degree in computer engineering from the Illinois Institute of Technology in 2020. His research interests include networks security, networks resource allocation, and ML-based networks optimization.

**Weiyi Zhang** (Senior Member, IEEE) is currently a Principal Inventive Scientist at AT&T Labs Research, Middletown, NJ, USA. Before joining AT&T Labs Research, he was an Assistant Professor with the Computer Science Department, North Dakota State University, Fargo, ND, USA, from 2007 to 2010. His research interests include reinforcement learning on networks planning and optimization, SDN and networks function virtualization for carrier networks, networks traffic demand forecast and analysis, and 5G wireless broadband strategic design. He has published more than 100 refereed papers in his research areas. He received the AT&T Labs Research Excellence Award in 2013, the Best Paper Award from IEEE GLOBECOM in 2007, the Best Paper Award from IEEE ICC in 2014, and the Best Paper Award from IEEE ICNP in 2017.

**Yu Cheng** (Senior Member, IEEE) received the B.E. and M.E. degrees in electronic engineering from Tsinghua University in 1995 and 1998, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Canada, in 2003. He is now a Full Professor with the Department of Electrical and Computer Engineering, Illinois Institute of Technology. His research interests include wireless networks performance analysis, information freshness, machine learning, networks security, and cloud computing. He was an IEEE ComSoc Distinguished Lecturer from 2016 to 2017. He received the Best Paper Award at QShine 2007 and the IEEE ICC 2011, and a Runner-Up Best Paper Award at ACM MobiHoc 2014. He received the National Science Foundation (NSF) CAREER Award in 2011 and the IIT Sigma Xi Research Award in the Junior Faculty Division in 2013. He has served as the Symposium Co-Chair for IEEE ICC and IEEE GLOBECOM; and the Technical Program Committee (TPC) Co-Chair for IEEE/CIC ICCC 2015, ICNC 2015, and WASA 2011. He was the Founding Vice Chair of the IEEE ComSoc Technical Subcommittee on Green Communications and Computing. He is an Associate Editor of IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE INTERNET OF THINGS JOURNAL, and IEEE WIRELESS COMMUNICATIONS.