Graph Neural Network-Based Node Deployment for Throughput Enhancement

Yifei Yang®, Dongmian Zou®, Member, IEEE, and Xiaofan He®, Senior Member, IEEE

Abstract—The recent rapid growth in mobile data traffic entails a pressing demand for improving the throughput of the underlying wireless communication networks. Network node deployment has been considered as an effective approach for throughput enhancement which, however, often leads to highly nontrivial nonconvex optimizations. Although convex-approximation-based solutions are considered in the literature, their approximation to the actual throughput may be loose and sometimes lead to unsatisfactory performance. With this consideration, in this article, we propose a novel graph neural network (GNN) method for the network node deployment problem. Specifically, we fit a GNN to the network throughput and use the gradients of this GNN to iteratively update the locations of the network nodes. Besides, we show that an expressive GNN has the capacity to approximate both the function value and the gradients of a multivariate permutation-invariant function, as a theoretic support to the proposed method. To further improve the throughput, we also study a hybrid node deployment method based on this approach. To train the desired GNN, we adopt a policy gradient algorithm to create datasets containing good training samples. Numerical experiments show that the proposed methods produce competitive results compared with the baselines.

Index Terms—Graph neural networks (GNNs), maximum flow, permutation invariance, reinforcement learning (RL), wireless communication networks.

I. INTRODUCTION

The increasing number of smart mobile terminals and devices drives a massive growth in mobile data traffic, which in turn leads to a pressing need for improving the throughput of the underlying wireless communication networks [1]. As the wireless channel and data rate between any two network nodes are often influenced by their locations, the throughput of many wireless networks, ranging from conventional cellular networks [2] and vehicular communication networks [3], to modern Internet of Things communication networks [4] and drone-enabled communication networks [5], closely hinges on the deployment of the network nodes.

In the literature, there have been some pioneering works devoted to improving the network node deployment. Some works mainly focus on optimizing the statistics and distributions of the network node locations [5], [6], [7], [8]. In these methods, network node locations are determined by brute-force search [5] or random selection [6], [7], [8], according to the obtained optimal node location statistics and distributions. Albeit their simplicity in design, the performances of these deployment strategies are often not satisfactory. To achieve better throughput, some other works consider the more fine-grained deployment strategy that aims to find the precise optimal location for each network node [9], [10], [11], [12]. However, the corresponding network deployment problems are much more involved and often nonconvex. To this end, approximation is commonly adopted to convert the original problem into convex ones. For instance, an alternative expression for the throughput with respect to node locations is obtained in [9] by successive convex approximation. However, this alternative term is only suitable for simple scenarios with a single relay node. For more general scenarios with multiple nodes, the original deployment optimization problem is transformed into an equivalent Lagrange dual problem and solved using the subgradient method in [10]. Another existing approach is to use the network eigenvalues derived from the spectral graph theory to approximate the network throughput [11], [12]. Albeit the above approaches make the node deployment optimization simpler, their approximation to the actual throughput may be loose, and hence sometimes lead to unsatisfactory performance.

To better address the throughput-optimal network node deployment problem, we propose a novel approach based on graph neural networks (GNNs), which offers a more precise approximation of throughput compared with the existing methods. In particular, we model the throughput as the maximum flow (max-flow) of a network and use a GNN to learn the functional relationship between the network max-flow and the node deployment. Then, we derive the gradients of the network max-flow with respect to node locations by conducting backpropagation in this GNN and update the node locations accordingly. Through in-depth analysis, we show that this method is theoretically reasonable by investigating the approximation capability of permutation-invariant neural networks, for which GNNs provide effective implementation. In addition, a hybrid approach that combines the existing spectral graph theoretic approach and the above GNN-based approach is proposed to further improve the performance.
Another challenge in training the desired GNN is to gather good training data. Ideally, we want to collect samples of the node locations that correspond to large network throughput. However, since the sample space (i.e., all possible node locations) in the considered problem is prohibitively large and continuous, it is intractable to collect sufficient representative data through random sampling. To this end, we propose a feasible approach to data collection that uses reinforcement learning (RL). Specifically, we apply a GNN-based proximal policy optimization (PPO), which has sufficient exploration capabilities, to search for good representative regions with an optimal max-flow to collect good training samples.

We summarize our main contributions as follows.

1) We propose a GNN-based node deployment method for network throughput enhancement. Based on this, a hybrid node deployment method is developed to further improve the throughput performance.
2) By proving that the class of permutation-invariant neural networks has the capacity for approximating both the function values and the gradients of multivariate permutation-invariant functions, we justify the correctness and effectiveness of the proposed method.
3) We propose to use a GNN-based PPO algorithm to collect good training data.

The remaining parts of the article are organized as follows. Section II briefly surveys related works. Section III formulates the problem and introduces a naive approach. In Section IV, we give theoretical justification and simple numerical examples. In Section V, we introduce the proposed methodology in detail. The experimental results are presented in Section VI. Eventually, Section VII concludes the article and states the limitations of our approach.

II. RELATED WORKS

A. Node Deployment for Wireless Networks

Recently, network node deployment has been widely used to address the challenges of wireless communication networks, including congestion mitigation [13], coverage maximization [14], and quality-of-experience optimization [15]. Network node deployment has also been considered for throughput enhancement. In [5], the space is first divided into grids and then the network deployment is implemented with a brute-force search method. In [6], a random geometry strategy based on the Poisson point process is developed to deploy the network nodes. In [7] and [8], uniform random deployment strategies are considered. Despite the simplicity of these deployment strategies in design, their performances are usually limited or suffer from intractable complexity.

For precise network deployment strategies, the requirement to find the exact locations of network nodes complicates the optimization problem. An analytical expression in [9] for the optimal location is derived by introducing the slack variables and leveraging the successive convex approximation. Based on the Lagrangian method, a dual problem for the approximate optimal solution is obtained in [10] by eliminating additional variables, which can be solved by a subgradient projection method. From a different viewpoint, the problem of maximizing the end-to-end throughput (i.e., max-flow) under jamming is considered in [11] and more recently in [12]. In both these works, the max-flow was approximated by the Cheeger constant and its maximization was done by a spectral graph theoretical approach. Nevertheless, the approximations in these works may be loose, and thus sometimes may lead to unsatisfactory performance.

B. Neural Networks in Wireless Communication

Recent years have witnessed wide applications of neural networks in wireless communications. For instance, neural networks can effectively model not only the nonlinear component of a wireless communication system, such as power amplifiers [16], [17], but also complicated signals transmitted over wireless channels [18]. Specifically, both [16] and [17] use the current and previous in-phase and quadrature phase \( I/Q \) components along with the envelope dependent terms to fit the \( I/Q \) components of the power amplifier output via neural networks. In [18], a neural-network-based method is developed to address the problem of state estimation based on signals transmitted over fading channels. Neural networks have also been used in wireless channel prediction and wireless sensor control. For instance, they are used for predicting channel characteristics in [19] and for building RL modules of sensor control in [20]. However, all these works mainly focus on the point-to-point communication scenarios with regular numerical data and cannot be directly applied to the more sophisticated scenarios of wireless networks that contain non-regular graph data; the rich edge feature information contained in the latter scenarios make GNN a more natural solution.

The recently advocated GNN has a significant impact on wireless communications [21], as many wireless networks can be naturally abstracted into graphs. In particular, GNNs have been widely applied to optimization problems in wireless communication networks [22], [23], [24], given the advantage that they can effectively use structural and topological information. These existing works mainly exploit permutation invariance and equivariance of GNNs to address the challenges in wireless communication networks. Wireless networks often carry crucial edge information, which can be effectively embedded in GNN representations. For instance, the edge features used in [22], [23], and [24] are information such as channel response, channel state, and equivalent channel. Different from the above works, the edge features in our work represent communication rates. Fortunately, many GNNs are capable of processing edge features [25], [26], [27], [28], [29], thanks to flexibility of using edge features in the design of the message passing scheme [30]. For a more general review of GNNs, the readers may refer to the recent comprehensive survey [31], which provides an extensive collection of resources on GNNs, encompassing advanced models and practical applications.

In addition to extracting features directly from the network, GNN can be embedded with deep RL to enable sufficient exploration capabilities for graph-structural data in wireless communication networks. For instance, a solution to the connection management problem based on GNN and deep Q-Learning algorithm is proposed in [32], which achieved...
C. Approximation Using GNN

GNNs have been applied to representation learning on graphs since the seminal work of [34]. The important properties of permutation equivariance and invariance make GNNs suitable for graph data since they do not depend on a specific ordering of nodes [35], [36], [37], [38]. However, due to these invariance properties, GNNs are not expected to enjoy universal approximation [39]. In fact, GNNs are not sufficiently expressive for representing all the permutation-invariant neural networks, and the equivalence between expressivity and approximation capability has been established in [40]. In particular, the Weisfeiler–Leman test is an important benchmark for the expressivity of GNNs, and recent studies have focused on going beyond such limits [27], [41], [42]. The universality of a specific class of invariant-equivariant GNNs is proved in [37], relying on a new generalized Stone–Weierstrass theorem for the algebra of real-valued equivariant functions. The more general notion of universal approximation of group-invariant networks is studied in [43].

Many applications use the gradients of neural networks to approximate those of the original functions. An early study numerically studies training neural networks to approximate a multivariate function and its first and second partial derivatives [44]. When both the function and gradient values are available, it is also possible to consider Sobolev training where the loss function is inherited from the Sobolev spaces [45], [46], [47]. However, Sobolev training is not applicable to our problem, since it is difficult to obtain precise gradients in our dataset.

III. PROBLEM FORMULATION AND A NAIVE SOLUTION

A. Problem Formulation

In this work, we consider a wireless communication network with $n$ nodes, as shown in Fig. 1. The node deployment is denoted by $(\xi_i, x_i) \in \mathbb{R}^2: i = 1, \ldots, n$, where $\xi_i$ and $x_i$ are the locations of a pair of stationary source and destination nodes, respectively, and $(\xi_i)_{i=2}^{n-1}$ are the locations of mobile relay nodes. With a slight abuse of notation, when there is no confusion, we may also use $\{\xi_i\}_{i=1}^{n}$ to denote the nodes themselves. We model this wireless communication network using a simple graph $G$ specified as follows. The vertex set of $G$, denoted by $\mathcal{V}$, consists of all the nodes $\{\xi_i\}_{i=1}^{n}$. The edge set of $G$, denoted by $\mathcal{E}$, consists of all the pairs of nodes. We assume that for a pair of nodes $\xi_i$ and $\xi_j$, the corresponding communication rate between the pair of nodes is a function of their locations and is denoted by $f_{\text{cap}}(\xi_i, \xi_j)$. Moreover, we assume $f_{\text{cap}}(\xi_i, \xi_j)$ is a Lipschitz continuous function with respect to $\xi_i$ and $\xi_j$, which holds in most wireless scenarios. Accordingly, we construct the weighted adjacency matrix $A$ of this network, whose $(i, j)$th entry $A_{i,j}$ is defined as the communication rate between nodes $i$ and $j$. That is,

$$A_{i,j} \triangleq \begin{cases} f_{\text{cap}}(\xi_i, \xi_j), & i \neq j; \\ 0, & i = j. \end{cases}$$

(1)

Our objective is to find the optimal locations of the relay nodes so that throughput (i.e., max-flow) between the source node and the destination node is optimized. It is well-known [48] that when the locations of the nodes are specified, this max-flow equals to the min-cut of the underlying network graph, defined by

$$C = \min_{\{S, \bar{S}\}} \sum_{i \in S, j \in \bar{S}} A_{i,j}$$

(2)

where $S$ is a subset of $\mathcal{V}$, $\bar{S}$ is the complement of $S$, and the minimum is taken over all the partitions $\{S, \bar{S}\}$ of $\mathcal{V}$.

Clearly, the maximum network flow $C$ is a function of the edge weights $A_{i,j}$’s, and thus also a function of the locations $\xi_i$ of the nodes. Actually, $C$ is a 1-Lipschitz function with respect to each edge weight $A_{i,j}$. We state this fact as a lemma.

Lemma 1: $C$ is a 1-Lipschitz function of $A_{i,j}$ for each $i, j \in \{1, \ldots, n\}$.

Proof: This simply follows from the fact that the change in $C$ cannot exceed the perturbation of $A_{i,j}$ because of the summation in (2). More specifically, we proceed as follows. Fix two different indices $i_0$ and $j_0$. For the adjacency matrix $A$, let $S^*$ and $\bar{S}^*$ be a partition of $\mathcal{V}$ with which the minimum in (2) is achieved, that is, $C = \sum_{\xi_i \in S^*, \xi_j \in \bar{S}^*} A_{i,j}$. Consider the perturbed adjacency matrix $\tilde{A}$ where $A_{i,j} = A_{i,j}$ if $(i, j) \neq (i_0, j_0)$ and $\tilde{A}_{i_0,j_0} \neq A_{i_0,j_0}$. Let $\tilde{C}$ denote the max-flow corresponding to $\tilde{A}$. If $A_{i_0,j_0} > \tilde{A}_{i_0,j_0}$, it is clear that

$$\tilde{C} = \min_{\{S, \bar{S}\}} \sum_{i \in S, j \in \bar{S}} \tilde{A}_{i,j} \leq \sum_{i \in S^*, j \in \bar{S}^*} A_{i,j} = C - \tilde{A}_{i_0,j_0} + A_{i_0,j_0}$$

and thus

$$|\tilde{C} - C| = |C - \tilde{C}| \leq \sum_{\xi_i \in S^*, \xi_j \in \bar{S}^*} A_{i,j} = |A_{i_0,j_0} - \tilde{A}_{i_0,j_0}|.$$

(4)

On the other hand, if $\tilde{A}_{i_0,j_0} < A_{i_0,j_0}$, then we consider $\tilde{A}$ as a perturbed version of $A$. For the same reason as above

$$|\tilde{C} - C| \leq |A_{i_0,j_0} - \tilde{A}_{i_0,j_0}|.$$

(5)
Combining both the cases, we conclude that $C$ is a 1-Lipschitz function of $A_i j$.

As a composition of Lipschitz functions, $C$ itself is Lipschitz with respect to the locations $\xi_i$ of the nodes as well (see e.g., [49, Lemma 2.1]). Accordingly, Rademacher’s theorem [50, Th. 3.1.6] immediately yields the following result.

**Lemma 2:** For $i = 2, \ldots, n - 1$, $C$ is a differentiable function of $\xi_i$ almost everywhere.

With the above notations, to optimize the throughput, we need to resolve the following problem.

**Problem 1:** For fixed source $\xi_j$ and destination $\xi_n$, find

$$\xi_2^*, \ldots, \xi_{n-1}^* = \arg \max \ C$$

(6)

where the maximum is taken over a designated domain $D$ (i.e., the deployable region of the relay nodes) of $\xi_2, \ldots, \xi_{n-1}$.

We remark that this problem is permutation-invariant. That is, if $(\xi_2^*, \ldots, \xi_{n-1}^*)$ is an optimizer of (6), then its any permuted version $((\xi_{\gamma(2)}, \ldots, \xi_{\gamma(n-1)}))$, where $\gamma$ is a bijection so that $\{\gamma(2), \ldots, \gamma(n-1)\} = \{2, \ldots, n-1\}$, is also an optimizer.

To solve Problem 1, we use the following iterative strategy. Start with certain positions $\xi_2[0], \ldots, \xi_{n-1}[0]$ and update the relay nodes along ascending directions (e.g., the directions of gradients) of the objective function $C$ with a small step-size $\varepsilon$. The node locations after $t$ updates are denoted by $\xi_2[t], \ldots, \xi_{n-1}[t]$, for $t = 1, \ldots, T$, where $T$ is the step count. These directions will be provided by a GNN as illustrated below.

To construct the corresponding GNN, we consider the graph $G = G(V, E, A)$. For each node, we define its features to be 3-D vectors $x_i = (x_i^{(0)}, x_i^{(1)}, x_i^{(2)}), i = 1, \ldots, n$. Here, the first coordinate $x_i^{(0)} = 1$ if the node is either the source or the destination (i.e., when $i = 1$ or $i = n$) and $x_i^{(0)} = 0$ otherwise. The coordinates $(x_i^{(1)}, x_i^{(2)})$ are exactly the location $\xi_i$ of the $i$th node. We define an $n \times 3$ matrix $X$, whose $i$th row is given by $x_i$, as a compact representation of the node features.

With the above notations, all the information of the wireless communication network is abstracted into the attributed graph $G$. We use $G$ and $X$ as the inputs of the GNN. Throughout this article, we use $g$, a function of both $X$ and $A$, to denote the functional expression of a GNN.

**B. Naïve Approach: Gradient Learning (GL) With GNN**

Since the gradients of $C$ with respect to the locations $\xi_2, \ldots, \xi_{n-1}$ exist almost everywhere by Lemma 2 and provide a direction of increasing $C$, a naïve approach is to fit these gradients with a GNN and then update the relay nodes according to the output (i.e., the estimation of the gradients) provided by this GNN. However, the desired training data of this GNN are not readily available, as it is quite challenging to directly compute the gradients of $C$ based on the combinatorial expression (2). For this reason, instead of finding the exact gradients, we train this GNN to find unit vectors $u_2, \ldots, u_{n-1}$ that lead to increment of $C$ and update the node locations from $(\xi_2, \ldots, \xi_{n-1})$ to $(\xi_2 + \xi u_2, \ldots, \xi_{n-1} + \xi u_{n-1})$ at each iteration. As discussed in Section V-C, these unit vectors can be found via RL with sufficient exploration capabilities.

Therefore, the problem of learning gradients can be treated as a node-level regression task, to be solved by the GNN $g$. Specifically, this GNN outputs an $(n - 2) \times 2$ matrix $g(X, A)$, whose rows are normalized to obtain the aforementioned unit vectors as follows:

$$\mathbf{Y} = [\mathbf{u}_i]_{i=2}^{n-1} = \left[ \frac{g_i(X, A)}{\|g_i(X, A)\|} \right]_{i=2}^{n-1}$$

(7)

where $u_i$ is the $(i - 1)$th row of the $(n - 2) \times 2$ matrix $Y$ and $g_i(X, A)$ is the $(i - 1)$th row of $g(X, A)$, respectively. We supervise the training of GNN by minimizing the mean squared error (MSE) $\|\mathbf{Y} - \hat{\mathbf{Y}}\|_F^2$, where $\mathbf{Y}$ is the matrix of the unit vectors provided by the training data obtained from RL. After this GNN is trained, in the test phase, at each step $t = 0, \ldots, T - 1$, the relay node locations are updated as follows:

$$\xi_i[t+1] = \xi_i[t] + \xi u_{i-1}[t], \quad i = 2, \ldots, n - 1.$$  

(8)

In (8), $u_{i-1}[t]$ is the $(i - 1)$th row of $\mathbf{Y}[t]$, which is the normalized version of $g(X[t], A[t])$. An overview of this procedure is shown in Fig. 2.

**Fig. 2.** Framework of GL. The red box illustrates the training phase, which learns the gradients with GNN; the blue box illustrates the testing phase, which updates the relay nodes to maximize the max-flow.

**IV. Theoretic Basis**

As will be shown in Section VI-D1, the naïve approach of GL still cannot deliver satisfactory performance. To address this issue, we will describe an alternative approach later in Section V. In particular, note that the value of $C$ can be efficiently computed by the Ford–Fulkerson algorithm [48]. This motivates us to directly estimate the value of $C$ by a GNN, and then we use the gradients of this GNN to approximate the gradients of $C$ and update the relay nodes accordingly.

Before presenting the details, some theoretic analysis is provided to validate the feasibility of this alternative approach. Since $C$ is permutation-invariant with respect to the node indices, we can achieve an arbitrarily small error of approximating $C$ using an expressive GNN, due to its capability of universal approximation [40], which is a variant of the multilayer perceptron (MLP) version [51] modulo equivariance.\footnote{Even if we have a good approximation of the function, its gradient is not necessarily close to that of the original function in general. For instance, let $f(x) = e^x$ and $f(x) = e^x + \sin(x)$ where $c$ is a constant. Clearly, $\|f - f\|_\infty = \nu$ while $\|f' - f'\|_\infty = \nu$. If we take a small $\nu$ but a large $\nu$, the gap between the function values will be small, but the norm of the difference between the gradients can be arbitrarily large.}
A. Theoretic Analysis

We first state the following proposition on both the function and gradient approximations for permutation-invariant ReLU neural networks.

**Proposition 1:** Let $f : [0, 1]^n \to \mathbb{R}$ be a differentiable permutation-invariant function such that $\nabla f$ is $L$-Lipschitz. Then for any $\epsilon, \delta > 0$, there exists an permutation-invariant ReLU neural network $g : [0, 1]^n \to \mathbb{R}$ for which both $|f(x) - g(x)| \leq \epsilon$ and $\|\nabla f(x) - \nabla g(x)\| \leq \epsilon$ hold for any $x \in [0, 1]^n - \mathbb{S}$ where $\mathbb{S}$ is a subset of $[0, 1]^n$ with Lebesgue measure $m(\mathbb{S}) < \delta$.

**Proof:** We first construct a continuous piecewise linear function that approximates $f$. For some $N \in \mathbb{N}$ that we will determine later, divide $[0, 1]^n$ into $N^n$ boxes of equal size, denoted by $\mathbb{B}_{i_1, \ldots, i_n}$, $i_1, \ldots, i_n \in \{1, \ldots, N\}$, where

$$
\mathbb{B}_{i_1, \ldots, i_n} = \left\{ x \in [0, 1]^n : \frac{i_l - 1 - \omega}{N} \leq x_l \leq \frac{i_l - \omega}{N}, \quad l = 1, \ldots, n \right\}.
$$

We also define $\tilde{\mathbb{B}}_{i_1, \ldots, i_n}$ for $i_1, \ldots, i_n \in \{1, \ldots, N\}$ by

$$
\tilde{\mathbb{B}}_{i_1, \ldots, i_n} = \left\{ x \in [0, 1]^n : \frac{i_l - 1 + \omega}{N} \leq x_l \leq \frac{i_l - \omega}{N}, \quad l = 1, \ldots, n \right\}
$$

for some $\omega > 0$ that we determine later. We denote the centers of the boxes $\mathbb{B}_{i_1, \ldots, i_n}$, which at the same time are the centers of $\tilde{\mathbb{B}}_{i_1, \ldots, i_n}$, to be $c_{i_1, \ldots, i_n} = ((i_1 - 1/2)/N, \ldots, (i_n - 1/2)/N)$.

Let $h : [0, 1]^n \to \mathbb{R}$ be defined by

$$
h(x) = \begin{cases} 
\tilde{h}_{i_1, \ldots, i_n}(x), & \text{if } x \in \tilde{\mathbb{B}}_{i_1, \ldots, i_n} \\
0, & \text{otherwise}
\end{cases}
$$

(9)

where

$$
\tilde{h}_{i_1, \ldots, i_n}(x) = f(c_{i_1, \ldots, i_n}) + \nabla f(c_{i_1, \ldots, i_n})^T (x - c_{i_1, \ldots, i_n}).
$$

(10)

Clearly, $\tilde{h}$ is linear on $\tilde{\mathbb{B}}_{i_1, \ldots, i_n}$ and permutation-invariant. Moreover, there exists a piecewise linear function $h$ such that $h = \tilde{h}$ on $\tilde{\mathbb{B}}_{i_1, \ldots, i_n}$, for instance, by interpolating the vertex values using nodal basis functions [52, Ch. 3] on $\mathbb{B}_{i_1, \ldots, i_n} - \tilde{\mathbb{B}}_{i_1, \ldots, i_n}$.

Although $h$ is not necessarily permutation-invariant, we construct a function $g : [0, 1]^n \to \mathbb{R}$ such that

$$
g(x) = \frac{1}{n!} \sum_{x' \in S(x)} h(x')
$$

(11)

where $S(x)$ is the set of all the permutations of $x$. Clearly, $g$ is permutation-invariant and its value agrees with $h$ on each $\mathbb{B}_{i_1, \ldots, i_n}$ since $h$ is permutation-invariant on $\tilde{\mathbb{B}}_{i_1, \ldots, i_n}$. Moreover, on each $\tilde{\mathbb{B}}_{i_1, \ldots, i_n}$, we have on one hand

$$
\|f(x) - g(x)\| \leq \|\nabla f\| \|x - c_{i_1, \ldots, i_n}\|
\leq M \frac{\sqrt{n}}{N}
$$

(12)

where $M$ is a uniform upper bound of $\nabla f$ on $[0, 1]^n$, which is finite due to the compactness of $[0, 1]^n$. On the other hand

$$
\|\nabla f(x) - \nabla g(x)\| = \|\nabla f(x) - \nabla f(c_{i_1, \ldots, i_n})\|
\leq L \|x - c_{i_1, \ldots, i_n}\| \leq L \frac{\sqrt{n}}{N}.
$$

(13)

Given $\epsilon, \delta > 0$, we choose $N = \lceil \epsilon/(\min[M, L] \sqrt{n}) \rceil + 1$. We also choose $\omega$ so small that $m(S) < \delta$ where

$$
S = [0, 1]^n - \bigcup_{i_1, \ldots, i_n = 1}^N \mathbb{B}_{i_1, \ldots, i_n}.
$$

(14)

Finally, by [53, Th. 5.2], $g$ can be implemented by a ReLU neural network with $\lceil \log_2(n + 1) \rceil$ layers.

We remark that the domain $[0, 1]^n$ of $f$ can be replaced by any compact permutation-invariant domain, which is the case in the considered node deployment problem.

Proposition 1 indicates that there exists a GNN that simultaneously satisfies the following: first, the GNN value is a close approximation of the function value of $C$ in (2); second, the gradients of this GNN well approximate the gradient of $C$ since GNN can be a practical implementation of the permutation-invariant neural network $g$. In GNN, permutation-invariance is guaranteed using graph convolutional layers, also known as message passing layers. In general, a graph convolutional layer can be represented as

$$
x'_i = y_\theta(x_j, \text{AGG}_{j \in \mathcal{N}(i)} \phi_\theta(x_j, x_j, A_{ij}))
$$

(15)

where $\mathcal{N}(i)$ denotes the neighbors of $i$, AGG is a permutation-aggregation function (e.g., sum), and $y_\theta$ and $\phi_\theta$ denote differentiable functions such as fully connected networks. Graph convolutional layers in the form of (15) all ensure permutation equivariance, but have different expressivity, and thus different approximation power. It is important to take a more expressive GNN to guarantee that a wide range of permutation-invariant neural networks can be represented, as studied in [40], [54], and [55]. To this end, we adopt the graph convolutional layer from [27] (denoted as GraphConv), which is a higher order GNN that considers multiscale graph structures and proves more expressive than the Weisfeiler–Leman test. Specifically, the message passing operation in GraphConv is represented as

$$
x'_i = W_1 x_i + W_2 \sum_{j \in \mathcal{N}(i)} A_{ij} \cdot x_j
$$

(16)

where $W_1$ and $W_2$ are matrices whose entries are trainable. We will also compare the performance with a less expressive GNN in ablation studies to validate this choice. On the other hand, to facilitate differentiation, especially for approximation of the gradients, we use the Gaussian error linear unit (GELU) [56] as the nonlinear activation function instead of ReLU in our experiments. We remark that GELU is very close to ReLU in value but much smoother.

B. Numerical Example

In this section, we will show synthetic numerical examples in which we use GNN to approximate multivariate functions and study their gradients.

We consider a simple fully connected graph with three nodes, associated with an input features matrix $X \in \mathbb{R}^{3 \times 2}$ and a scalar output $y \in \mathbb{R}$. We assume that the true value of $y$ is an analytical function of $X$. Specifically, we consider the following two functions:

$$
f_1(X) = x_{11}^2 + x_{12}^2 + x_{21}^2 + x_{22}^2 + x_{31}^2 + x_{32}^2,
\quad x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32} \in (1, 4)
$$

(17)
We then train a GNN with a cascading of the following layers.

1) Input Layer: 2-D.
2) $3 \times$ GraphConv Layers: 32-D.
3) Pooling Layers: Global addition pooling (Global-add-pool).
4) $2 \times$ Linear Layers: 32-D.
5) Output Layer: 1-D.

Between the GraphConv layers, we use the GELU activation function and the GraphSizeNorm [57] where we apply normalization over each individual graph in a batch of node features. Moreover, the activation function between the linear layers is also GELU. For each setting, we fit the GNN to the function value with a learning rate of 0.002, and a batch size of 100, for 1000 epochs.

Suppose the trained GNN is denoted by $g$. For each testing example $X$, we compute both $g(X)$, the output value of GNN, and $Vg(X)$, the gradients of GNN with respect to the input variables. To evaluate the performance of GNN, we use the maximum relative error, defined as

$$\max_{X} \frac{|g(X) - y|}{|y|}$$

for the function value and

$$\max_{X} \frac{\left| \frac{\partial g(X)}{\partial x_{k}} - \frac{\partial y}{\partial x_{k}} \right|}{\left| \frac{\partial y}{\partial x_{k}} \right|}, \quad k, l \in \{1, 2, 3\}$$

for the gradients, where the maximum is taken over the collection of the testing examples in each case.

The experimental results are shown in Table I, where we observe consistently small errors from the partial derivatives, albeit not included in the loss functions. For a more detailed discussion, we also show the distributions of the relative errors of function value and first-order partial derivatives in Fig. 3. For $f_1$, the maximum relative error is 0.11%, but most of the relative errors of function value are within 0.05%. The maximum relative error of the first-order partial derivatives is 6.15%, but 99.03% of them are within 2%. On the other hand, for $f_2$, the maximum relative error is 0.24%, but 96% of the relative errors of function value are within 0.04%. The maximum relative error of the first-order partial derivatives is 6.03%, but 98.77% of them are within 1.5%.

From these experimental results, we see that after fitting GNN to the function value, it can not only faithfully represent a multivariable function but also well approximate its first-order partial derivatives.

| Table I: Maximum Relative Errors Between GNN and Function Outputs |
|----------------------|------------------|
| $i$ | maximum relative error |
| $f_1$ | 0.11% | 0.24% |
| $\partial x_{11}f_1$ | 1.32% | 2.15% |
| $\partial x_{12}f_1$ | 2.82% | 1.58% |
| $\partial x_{21}f_1$ | 3.04% | 1.83% |
| $\partial x_{22}f_1$ | 6.15% | 1.74% |
| $\partial x_{11}f_2$ | 3.76% | 6.03% |
| $\partial x_{12}f_2$ | 2.07% | 5.89% |

For each $f_i$, $i = 1, 2$, we take a collection of 30,000 training examples of pairs $(X, f_i(X))$. In each training example, we randomly sample $x_{11}, x_{12}, x_{21}, x_{22}, x_{31}, x_{32}$ independent of the uniform distribution $\text{Unif}(1, 4)$ and then compute $y = f_i(X)$. We also collect 500 testing examples in the same manner. We then train a GNN with a cascading of the following layers.

1) Input Layer: 2-D.
2) $3 \times$ GraphConv Layers: 32-D.
3) Pooling Layers: Global addition pooling (Global-add-pool).
4) $2 \times$ Linear Layers: 32-D.
5) Output Layer: 1-D.

Motivated by the above theoretic result, we propose directly fitting a GNN to the max-flow of the network and determine the update directions of the relay nodes by differentiating the GNN, as shown in Fig. 4. Unlike the naive method of learning gradients, the learning objective of this approach is the scalar max-flow value of the network, which replaces the original node-level task to a graph-level task.

After we fit the GNN to the max-flow value, the test phase is conducted as follows. First, the graph representations $G(V, E, A)$ of test node locations are fed into the trained GNN. Then, we perform autodifferentiation to obtain the gradients of the output with respect to both the node features $X$ and the weighted adjacency matrix $A$

$$\frac{\partial Vg(X, A)}{\partial x_{i}^{(m)}} = \left( \frac{\partial g(X, A)}{\partial X} \cdot \frac{\partial X}{\partial x_{i}^{(m)}} \right) + \left( \frac{\partial g(X, A)}{\partial A} \cdot \frac{\partial A}{\partial x_{i}^{(m)}} \right)$$

(21)

where $\langle \cdot, \cdot \rangle$ is the Hilbert–Schmidt inner product and

$$\frac{\partial A}{\partial x_{i}^{(m)}} = \left[ \frac{\partial A_{p,q}}{\partial x_{i}^{(m)}} \right]_{p,q=1}^{n}$$

(22)
Here, \((\partial A_{p,q})/(\partial x_i^{(m)})\) depends on the network of interest and an example is given in Section VI-A. Finally, the relay nodes are updated along these directions with stepsize \(\xi\) as follows:

\[
x_i^{(m)}[t+1] \leftarrow x_i^{(m)}[t] + \xi \frac{\partial_i}{\partial x_i^{(m)}} g(X[t], A[t]), \quad m = 1, 2
\]

where \(\partial_i = \partial g/\partial x_i^{(m)}\).

### B. Hybrid Method

To further improve the performance of throughput optimization, we also study a hybrid method based on an existing method [11] and the proposed MFL method.

1) Existing Method: In the spectral graph theory, the Cheeger constant of the non-normalized Laplacian matrix \(L\) is an approximation to the max-flow \(C\) [58]. This Cheeger constant can be bounded by an inequality of the second smallest eigenvalue \(\lambda_2(L)\) of the Laplacian matrix, which depends on the locations of the nodes [59]. To better capture the max-flow between the source node and the destination node [11], a weighted Cheeger constant (WCC) is used in the approximation of \(C\). Specifically, let \(W\) be the diagonal weight matrix, one defines the weighted Laplacian matrix to be \(L_W = W^{-1/2}LW^{-1/2}\). The relay nodes can then be updated to maximize the second small eigenvalue, \(\lambda_2(L_W)\), of the weighted Laplacian matrix to optimize \(C\).

2) Hybrid Method: The hybrid method adopts a greedy algorithm that chooses the one with larger increment of max-flow between the GNN-based MFL method and the WCC method in each step, as shown in Fig. 5. In particular, two different sets of gradients (i.e., the node update direction corresponding to the max-flow increase), Gradients 1 and 2, are obtained using the GNN-based MFL and WCC methods, respectively. Two candidate deployments, Deployment 1 and 2, can be obtained according to these gradients. By comparison, the set of gradients that leads to a higher max-flow will be selected and the corresponding candidate deployment will be taken as the new deployment. We keep iterating until a prespecified number of steps is reached. This hybrid method is expected to synthesize the advantages of both the GNN-based MFL method and the WCC method.

C. Training Dataset Creation

To train the GNN to learn the value of \(C\) (or, equivalently, the max-flow), it is vital to create a dataset that well represents max-flow values for different node locations. In particular, it is important that the training dataset contains some node locations corresponding to good max-flow values, so that the GNN will not underestimate the maximum possible value of the max-flow of the network. To this end, we propose to use an RL model based on a GNN implementation of PPO [60], [61] to obtain suitable training samples. This algorithm has a strong exploration capability to find the node locations corresponding to some good max-flow values in the sampling space.

In our context, the three key elements of the PPO algorithm are defined as follows.

1) State (s): The graph-structured data including node features \(X\) and edge features \(A\).
2) Action (a): A collection of 2-D vectors representing the update directions of the relay nodes.
3) Reward (r): The increment of the max-flow by taking an action at a given state.

The agent in the PPO algorithm consists of an actor (i.e., the policy) and a critic (i.e., the value function), both modeled as GNNs. The actor GNN provides a policy for choosing the next action based on the current state. The critic GNN uses the policy to estimate the value of the state-action pair, which is used to update the policy through backpropagation.

We remark that PPO could also be used for optimizing the max-flow directly, but we do not expect a good performance due to the complexity of the search space (see Section VI-D2 for a comparison of performance). Therefore, the application of PPO is only regarded as a pretraining step which provides us with a good variety of training data.
an action for a given state, while the critic GNN evaluates possible cumulative rewards for a state. In particular, given a current state as the input, the outputs of the actor GNN are used as means and variances of \((n - 2) \times 2\) normal distributions \(\text{Norm}(\mu_i, \sigma_i, i = 2, \ldots, n - 1, l = 1, 2)\). The update of the relay node \(\xi_i, i = 2, \ldots, n - 1\), is then a vector \((\Delta x_{i,1}, \Delta x_{i,2})\), sampled according to \(\Delta x_{i,1} \sim \text{Norm}(\mu_{i,1}, \sigma_{i,1})\) and \(\Delta x_{i,2} \sim \text{Norm}(\mu_{i,2}, \sigma_{i,2})\). On the other hand, the critic GNN outputs a deterministic scalar value as the cumulative rewards.

The PPO algorithm runs for \(N\) epochs, and each epoch is divided into \(T\) time slots. Here, \(T\) is equal to the product of the number of update steps \(T\) and the number of updates from scratch \(\varphi\). For the \(i\)th epoch, \(i = 1, 2, \ldots, N\), we use \(\pi_{\theta_i}\) to denote the actor GNN and \(V_{\phi_i}\) to denote the critic GNN, where \(\theta_i\) and \(\phi_i\) denote the parameters in these two GNNs, respectively. We also use \(s_i\), \(a_i\), \(r_i\) to denote the state, action, and reward at the \(t\)th time slot, \(t = 1, 2, \ldots, T\), respectively, where for brevity we omit their dependence on \(i\). Moreover, let \(\pi_{\theta_i}(a_i|s_i)\) denote the probability that the policy \(\pi_{\theta_i}\) executes \(a_i\) at a given \(s_i\), and \(s_i'\) denote the state after executing \(a_i\) at \(s_i\). With the above notations, we can represent the advantage \(A_i(\theta_i)\) as

\[
A_i(\theta_i) = R_i - V_{\phi_i}(s_i)
\]

where \(V_{\phi_i}(s_i)\) is the current value function from the critic GNN, and

\[
R_i = r_i + \gamma V_{\phi_i}(s_i').
\]

Here, \(\gamma\) is a hyperparameter. Accordingly, the loss function for updating the actor GNN is defined as

\[
L_{\text{actor}}(\theta) = \mathbb{E}\left[\min[\rho_{\theta}(\theta)A_i(\theta)]ight.\left.\text{clip}(\rho_{\theta}(\theta), 1 - \tau, 1 + \tau, A_i(\theta))\right]
\]

(26)

where \(\mathbb{E}[\cdot]\) is the empirical average over all the samples in the buffer (stored the training data of PPO), \(\tau\) is a hyperparameter, and \(\text{clip}(\cdot, \cdot, \cdot, \cdot)\) is a clipping function defined as

\[
\text{clip}(a, b, c, d) = \begin{cases} 
\min\{a, c\} \cdot d, & \text{if } d > 0 \\
\max\{a, b\} \cdot d, & \text{otherwise}
\end{cases}
\]

(27)

and \(\rho_i\) is a quotient of probability densities defined as

\[
\rho_i(\theta) = \frac{\pi_{\theta}(a_i|s_i)}{\pi_{\theta_i}(a_i|s_i)}.
\]

(28)

On the other hand, the loss function for updating the critic GNN is defined as

\[
L_{\text{critic}}(\phi) = \mathbb{E}\left[\mathcal{L}(V_{\phi_i}(s_i) - R_i)\right]
\]

(29)

where \(\mathcal{L}\) is a piecewise function on \(\mathbb{R}\) defined as

\[
\mathcal{L}(x) = \begin{cases} 
0.5|x|^2, & \text{if } |x| < 1 \\
|x| - 0.5, & \text{otherwise.}
\end{cases}
\]

(30)

For clarity, we summarize the above procedures in Algorithm 1 and Fig. 6. Based on this framework, we explore the state space in a comprehensive way so that a sufficiently rich collection of update trajectories of the relay nodes can be obtained. Each trajectory \(\{(\xi_{1,t}], \xi_{2,t], \ldots, \xi_{n,t}]\}, t = 0, \ldots, T\) consists of the positions of legitimate nodes that are updated \(T\) times. To construct the training dataset, we collect data points \(\{(\xi_{1,t], \xi_{2,t], \ldots, \xi_{n,t}]}, t = 0, \ldots, T\}\) along each trajectory at a fixed time interval \(k\).

### VI. Experiments

#### A. Experimental Setup

We test our methodology in the scenario where, in addition to a wireless network with the source, destination, and relay nodes (which will be referred to as legitimate nodes altogether), there also exists a jammer. Note that similar anti-jamming communication scenarios are commonly considered in the wireless communication literature [9], [11], [12], [62]. In the considered scenario, the jammer is assumed to be positioned at a known location \(\xi_{j} = (x_{j}^{(1)}, x_{j}^{(2)}) \in \mathbb{R}^2\) and it interferes the data transmission within the wireless network. The signal-to-interference ratio (SIR) for the transmission from node \(i\) to node \(j\), \(\text{SIR}_{i,j}, i,j \in \{1, \ldots, n\}\) can be modeled as

\[
\text{SIR}_{i,j} = \frac{d^{-\alpha}_{i,j}}{\eta d_{j,k}^{-\alpha} + \sum_{k \notin \Xi_{i,j}} v(d_{j,k}/r_{\text{int}})}
\]

(31)

where \(d_{i,j} = ||\xi_i - \xi_j||\) is the distance between node \(i\) and node \(j\), \(d_{j,k} = ||\xi_j - \xi_k||\) is the distance between node \(j\) and the jammer, and \(\eta\) is the power ratio of the jammer to the legitimate nodes. Note that a path-loss model [63] for the wireless channel is assumed in (31) and \(\alpha\) is the path loss factor, and \(r_{\text{int}}\) is a design parameter. Also, \(\Xi_{i,j} = \{1, \ldots, n\}\backslash\{i, j\}\), where “\’\’\’ denotes set difference, is the set
of indices of legitimate nodes excluding $i$ and $j$. To prevent violation of physical constraints, $v$ is defined by

$$v(z) \triangleq \frac{\exp(-\kappa z - \log z_0)}{1 + \exp(-\kappa z - \log z_0)}$$

(32)

where $\varrho$ and $\kappa$ are all design parameters, and $z_0$ can be an arbitrarily small positive number. Based on the above model, the (two-way average) communication rate $f_{\text{cap}}$ in (1) between two nodes $i$ and $j$ is defined as

$$f_{\text{cap}}(\xi_i, \xi_j) \triangleq \frac{B}{\ln(1 + \text{SIR}_{i,j})^{-1} + \ln(1 + \text{SIR}_{j,i})^{-1}}$$

(33)

where it is assumed that orthogonal spectrum is allocated to each link of the network, and $B$ denotes the corresponding bandwidth per link. Therefore, the corresponding derivatives $(\partial A_{p,q})/(\partial x_i^{[m]})$ [see (22)] for this considered scenario can be found by the chain rule based on (31)–(33).

In our experiments, we consider a wireless network with $n = 6$ legitimate nodes in a $600 \times 600$ m² area. We represent this area by a Cartesian coordinate system, where the range of the $x$-coordinate is $(-6, 6)$ and the range of the $y$-coordinate is $(-6, 6)$, and the unit length in both the coordinates is set to 50 m. The coordinates of the source and the destination nodes are fixed to $\xi_1 = (-4.5, 0)$ and $\xi_6 = (4.5, 0)$, respectively. In each deployment, the relay nodes are initialized with the following coordinates: $\xi_2 = (-2.7, 0)$, $\xi_3 = (-0.9, 0)$, $\xi_4 = (0.9, 0)$, and $\xi_5 = (2.7, 0)$. The coordinates of the jammer nodes are randomly chosen with the following constraints: $x_j^{(1)} \in (-6, 6)$, $x_j^{(2)} \in (-6, 6)$, $d_{i,j} > 3$, and $d_{6,j} > 3$, where the last two constraints represent “guard zones” that prevent the jammer from getting too close to the source or destination.

To generate the training set, we initialize a total of 2000 deployments. Every deployment is updated for $T = 400$ steps with stepsize $\zeta = 0.02$ following the PPO strategy described in Section V-C. In the GNN-based PPO, we choose $\gamma = 0.9$, $\tau = 0.2$, $\varphi = 5$, $T = 400 \times 5 = 2000$, and $M = 10$. We collect a data point once every five steps from each deployment. For convenience, we name this dataset the RLGP (short for RL GNN-PPO) dataset. For efficiency, the actor GNN and the critic GNN stop training when the reward converges or the number of epochs reaches the maximum. On the other hand, the test set contains 500 initial deployments which have different jammer locations.

We used the PyTorch Geometric library [64] to implement all the GNN-related tasks.

### B. Baseline Methods

We compare MFL with the following baseline methods.

1) **Gradient Learning (GL):** This baseline is the naïve approach introduced in Section III-B. The GNNs in both MFL and GL are trained using the RLGP dataset described in Section VI-A. The label for MFL is the max-flow value, while the labels for GL are the directions of the relay nodes traveled in the training procedure.

2) **Reinforcement Learning (RL):** This baseline is a direct application of PPO to the test data. Specifically, we train an agent to explore the test data and provide the update directions for the relay nodes. The training procedure still follows the framework illustrated in Fig. 6. We obtain the optimal trajectory of each test deployment and use it to compare with MFL. To ensure good results, the number of timeslots, $T$, is chosen to be 40,000, and the number of epochs, $N$, is 10,000.

3) **Weighted Cheeger Constant (WCC):** This baseline is introduced in Section V-B1. As in the CC method, we use the gradients of $\lambda_2(L_w)$ to adjust the locations of relay nodes. In our experiments, the weight matrix is $W = \text{diag}[w_1, \ldots, w_n]$, where $w_i = 3n$ if $i = 1, n$, and $w_i = 1$ if $i \neq 1, n$. 

---

Fig. 6. GNN-based PPO framework. Step 1: exploration based on the current state $s_i$ and the current actor $\pi_\theta$ to obtain $\pi_\theta(a_i|s_i)$, from which $a_i$ is sampled. Following the update of $a_i$, we obtain $r_i$ and $s_{i+1} = s_i'$. These data are stored in the buffer until it is filled. Step 2: estimate $A_t$ according to (24) and $R_t$ according to (25) with all data in the buffer based on the current critic. Steps 3 and 4: based on the data in the buffer and the estimated $A_t$ and $R_t$, the actor and critic are updated multiple times according to (26) and (29), respectively.
C. GNN Architectures

In our experiments, we use several different GNNs. We describe their structures as follows.

1) GNN in GL: In GL, we use a GNN that consists of a cascading of three GraphConv layers, each followed by the GELU activation function. The outputs of the last GraphConv layer are then pushed forward through two linear layers, with another GELU activation in between. This architecture is illustrated in Fig. 7.

2) GNN in MFL: Since MFL is a graph-level task, we need to construct a permutation-invariant GNN through a global pooling layer. To this end, we take a global sum of the features of all the nodes which returns a graph-level output after all the GraphConv layers. We use again the GELU activation functions between layers. We illustrate this GNN in Fig. 8.

3) GNNs in GNN-Based PPO: The structures of the actor GNN and the critic GNN are shown in Fig. 9. The actor GNN consists of a cascading of the following: two GraphConv layers, each with a GELU activation function; global sort pooling; and two different fully connected linear layers, whose outputs are activated by tanh and sofplus = \log(1 + \exp(\cdot)), respectively. The critic GNN has the same graph convolutional layers and activation function, but we use a global addition pooling layer followed by a fully connected linear layer.

D. Comparison Results

We use the same test set described in Section VI-A to evaluate the performance of all the baseline methods mentioned above. In Fig. 10(a), we illustrate an example showing the trace of updates during the testing process, where the relay nodes are updated for T = 400 steps. The corresponding change in the max-flow value of this network is shown in Fig. 10(b). The performance of the methods is measured by the final max-flow of the network after the relay node location update.

1) Comparison With GL: We first compare the performance between MFL and GL. In Fig. 11(a), we summarize the relative difference in the final max-flow between the two methods in a histogram. Here, a positive number in the horizontal axis (">0") indicates that the MFL method is superior and a negative number ("<0") indicates the contrary. There are 494 deployments out of 500 in which MFL outperforms GL, and in 420 of them, the final max-flow value from MFL is at least 20% higher than GL. There are also a few deployments where MFL is significantly superior. We further illustrate the locations of the jammer and imply the better performing method in Fig. 11(b). Clearly, only when the jammer node is located near the origin may GL outperform MFL.

2) Comparison With RL: We compare MFL and RL and show the results in Fig. 12. As shown in Fig. 12(a), in 449
Fig. 11. Comparison with the GL method. (a) Histogram of the max-flow difference. (b) Distribution of jammer nodes: the blue circles indicate the locations of jammer nodes for which the MFL method excels, the pink triangles indicate the locations of jammer nodes for which the GL method excels, and the red circular dashed line indicates the boundary of the guard zones.

Fig. 12. Comparison with the RL method. (a) Histogram of the max-flow difference. (b) Distribution of jammer nodes: the blue circles indicate the locations of jammer nodes for which the MFL method excels, the green triangles indicate the locations of jammer nodes for which the RF method excels, and the red circular dashed line indicates the boundary of the guard zones.

Fig. 13. Comparison with the WCC method. (a) Histogram of the max-flow difference. (b) Distribution of jammer nodes: the blue circles indicate the locations of jammer nodes for which the MFL method excels, the red triangles indicate the locations of jammer nodes for which the WCC method excels, and the red circular dashed line indicates the boundary of the guard zones.

Fig. 14. Comparison with the MFL method and the WCC method. (a) Histogram of the max-flow difference between the hybrid method and the MFL method. (b) Distribution of jammer nodes: the blue circles indicate the locations of jammer nodes for which the MFL method excels, the yellow diamonds indicate the locations of jammer nodes for which the hybrid method excels, and the red circular dashed line indicates the boundary of the guard zones. (c) Histogram of the max-flow difference between the hybrid method and the WCC method. (d) Distribution of jammer nodes: the red triangles indicate the locations of jammer nodes for which the MFL method excels, the yellow diamonds indicate the locations of jammer nodes for which the hybrid method excels, and the red circular dashed line indicates the boundary of the guard zones.

(more than 89%) out of 500 deployments we observe a superior performance from MFL. Fig. 12(b) implies that the jammer nodes for which MFL performs worse are mainly located around the edge of the guard zones.

3) Comparison With WCC: Finally, we compare the performance of MFL and WCC and show the result in Fig. 13. As shown in Fig. 13(a), MFL outperforms WWC in 377 deployments (more than 75%) out of 500. From Fig. 13(b), we can observe again that MFL is less effective than WCC mainly at the edge of the guard zones.

4) Discussion: It is clear from the results that only for some special locations of the jammer node near the origin or the edge of the guard zones, MFL is not as effective as baseline methods. We believe the reason is that the jammer node in these deployments is too close to the relay nodes, which causes the SIR between the surrounding relay nodes to be close to zero. In such cases, a small change in locations of the relay nodes may lead to a dramatic change in the max-flow. Our GNN is smooth and may not be capable of representing such drastic changes.

E. Results of Hybrid Method

We present the experimental results of the hybrid method in Fig. 14. The performance of the hybrid method excels both individual approaches. Compared with the MFL method, the hybrid method is superior in 418 test deployments. Fig. 14(a) reveals that although most of the data only see an improvement of 20% or less, there were still some deployments with an improvement of 50% or more, with the largest reaching 641%. In Fig. 14(b), we observe that some deployments for which the MFL method does not perform well have an improved result when we use the hybrid method. On the other hand, as observed in Fig. 14(c), 94% of the test data enjoy improved results when the hybrid method replaces the WCC method. Although the hybrid method performs well, Fig. 14(d) shows clearly that there are still specific regions of the jammer node where the hybrid method falls behind the WCC method. We believe the hybrid method does not completely resolve the problem at the edge of guard zones, from which the MFL method suffers.

To summarize the results more clearly, we use the WCC method as a benchmark to quantitatively compare the average performance in the test set. We show the average difference in
the max-flow (avg. diff. mf.) and the average relative difference (avg. rel. diff. mf.) in Table III. In addition, to eliminate the influence of outliers, we calculate the truncated-averaged difference (c-avg. diff. mf.) and the truncated-averaged relative difference (c-avg. rel. diff. mf.), excluding the highest and lowest 10% values. In all these metrics, the hybrid method has the best overall performance.

### F. Ablation Studies

We perform ablation studies to show two important ideas implemented in our experiments. The first is the use of RL for generating the dataset and the second is the use of expressive GNN layers throughout our experiments. We then further validate the use of the GELU activation and the global addition pooling layer over alternative choices.

1) **Comparison of Different Datasets:** In addition to the RLGP dataset in our main experiments, we create two additional datasets, namely, the random walk (which we abbreviate as “RW”) dataset and the weighted Cheeger constant (which we abbreviate as “WCC”) dataset. The setups of generating two additional datasets are the same as RLGP, except for the strategy for moving the relay nodes. Specifically, the RW dataset takes a direction uniformly on the unit circle, while the WCC dataset follows the WCC method when choosing the direction. Both the datasets follow the same stepsize $\zeta$ when moving the relay nodes. We use the RW dataset, WCC dataset, and RLCP dataset to train GNN, respectively, and implement the MFL method with these GNNs. The comparison of their test results is shown in Fig. 15. In Fig. 15(a), the histogram reveals that the results using the RLGP dataset are remarkably superior to the RW dataset. About 472 of the results of the RLGP dataset are superior to the results for the RW dataset, and 211 of them have a final max-flow larger than 40% relative to the RW, with the largest even reaching 713%. From Fig. 15(b), it is clear that the results using the RLGP dataset excel the WCC method. Indeed, we observe that in 390 deployments, which is close to 80% of the total, the final max-flow values are higher when the GNN is trained using the RLGP dataset. We conclude that using the RLGP dataset makes a significant contribution to the excellent performance of our proposed methodology.

2) **Comparison of Different Model Components:**

   a) **Comparison of different graph convolution layers:** To show the effect of using expressive GNN layers, we compare using GraphConv with GINEConv and GCNConv, which are the convolutional layer used in graph isomorphism network [28] and graph convolutional network [29] respectively, with edge features. While GraphConv is a higher order method, GINEConv and GCNConv are both first-order, and thus less expressive and have more deficient approximation power. We apply GraphConv, GINEConv, and GCNConv to the GNN in the MFL method, respectively. The comparison results are shown in Fig. 16. As can be seen, in Fig. 16(a), GNN with GraphConv layers performs much better than GINEConv. About 478 (more than 95.6%) out of 500 deployments see a superior performance of GNN with GraphConv. In particular, in 402 of these deployments, GraphConv achieves a final max-flow at least 30% larger than GINEConv. In Fig. 16(b), for 73.4% of the test data, GraphConv performs better than GCNConv, while for 19.2% of the test data, GraphConv achieves an improvement of over 30%.

   b) **Comparison of different activation functions:** We demonstrate the advantage of using a smooth activation function by comparing the performance of GELU with ReLU and LeakyReLU. Our results in Fig. 17(a) indicate that 343 out of 500 deployments perform better with GELU over ReLU. About 478 (more than 95.6%) of 500 deployments see a superior performance of GNN with GraphConv. In particular, in 402 of these deployments, GraphConv achieves a final max-flow at least 30% larger than GINEConv. In Fig. 17(b), for 73.4% of the test data, GraphConv performs better than GCNConv, while for 19.2% of the test data, GraphConv achieves an improvement of over 30%.

   c) **Comparison of different global pooling layers:** We compare global-add-pool with global mean pooling (global-mean-pool) and global max pooling (global-max-pool), respectively, and illustrate the results in Fig. 18. The performance of the global-add-pool surpasses both global-mean-pool and

### Table III

|                  | avg. diff. mf | avg. rel. diff. mf | c-avg. diff. mf | c-avg. rel. diff. mf |
|------------------|--------------|--------------------|-----------------|----------------------|
| WCC              | 0            | 0                  | 0               | 0                    |
| GL               | -0.3099      | -35.47%            | -0.3016         | -33.79%              |
| RL               | -0.2196      | -27.82%            | -0.2188         | -27.53%              |
| MFL              | -0.0128      | -1.43%             | 0.0116          | 1.32%                |
| Hybrid           | 0.0314       | 4.21%              | 0.0301          | 3.49%                |
global-max-pool. Specifically, Fig. 18(a) shows that global-add-pool only slightly excels global-mean-pool. On the other hand, Fig. 18(b) shows that in 67.6% of the test deployments, global-add-pool performs better than global-max-pool. We conclude that both global-add-pool and global-mean-pool summarize the features effectively, while global-max-pool may lose important information contained in non-max values.

### G. Additional Results on Accuracy of MFL

In addition to the above experimental results, we compare the learned max-flow with the true values. We take 500 deployments where the network nodes are located at the ending positions determined by the hybrid method on the test dataset. We show the average relative error (avg. rel. err.) and the truncated-averaged relative error (c-avg. rel. err.) excluding the highest and lowest 1% values in Table IV. It can be seen that the GNN-based MFL can achieve a more precise approximation of the max-flow, which is significantly superior to the WCC method.

|                              | avg. rel. err. | c-avg. rel. err. |
|------------------------------|----------------|------------------|
| \( \lambda_2(L_W) \)        | 93.82%         | 93.86%           |
| upper bound of WCC          | 16.03%         | 14.03%           |
| lower bound of WCC          | 96.91%         | 96.93%           |
| output of MFL               | 1.55%          | 0.79%            |

### H. Sensitivity to Hyperparameters

In this section, we perform sensitivity analysis for the hyperparameters listed in Table II regarding GNN. Each time, we use an alternative value for one hyperparameter while keeping the others fixed. Specifically, we consider different learning rates (0.0001 and 0.0004), batch sizes (150 and 200), and numbers of neurons (64 and 128). Keeping the results reported for the MFL method as a baseline, we record the difference in the max-flows in Table V. The metrics for comparison are the same as in Table III. We observe that our model is not sensitive to the learning rate, the batch size, or the number of neurons in each layer.

### I. Discussion

We discuss two potential practical applications of our approach. As an example, during natural disasters such as earthquakes, mudslides, or floods, the time prior to rescue efforts is critical for protecting lives, properties, and national infrastructure. In such scenarios, the stability of communication in the rescue area is paramount for effective rescue operations. Conventional communication facilities in such circumstances are likely damaged, and drone-enabled communication networks are an ideal solution due to their high mobility and on-demand deployment. In this scenario, our approach can be applied to deploy the drones and establish an effective communication network for disaster management operations. Another possible exemplary application of our approach is to deploy mobile base stations to ensure dependable wireless communications during major events. Due to surge of communication demands in major events or unforeseen radio interference, communication outages may occur. In such circumstances, our approach can be leveraged to promptly deploy some small mobile base stations to strengthen the communication links and provide satisfactory communication services.

### VII. Conclusion

In this article, we studied the problem of throughput maximization by optimizing the node deployment of wireless communication networks. Specifically, we modeled the throughput as a network max-flow and used an expressive high-order GNN to learn the max-flow value under the various deployments of networks in a supervised fashion. In the test phase, the trained GNN provided gradients for updating the locations of the relay nodes. The correctness and effectiveness of the...
proposed approach were supported theoretically by investigating the approximation capabilities of permutation-invariant neural networks. By searching the effective dataset with RL, we achieved competitive results compared with baselines. Combining the existing spectral graph theory approach and the GNN-based approach, the proposed hybrid method provided some improvement in performance. We believe that our method has a wide spectrum of potential applications such as unmanned aerial vehicle communication networks, mobile ad hoc networks, and robotic sensor networks.

There are some limitations to the current method. In particular, this GNN-based method does not perform as well as other methods in some special deployments, which also cannot be completely solved by the hybrid method. We hypothesize the reason is that in those deployments, the max-flow, as a function of locations, is not smooth enough and the generalizability of the underlying GNN model will deteriorate. In the future, we will look for better approximation models to handle these cases.

REFERENCES

[1] W. H. Chin, Z. Fan, and R. Haines, “Emerging technologies and research challenges for 5G wireless networks,” IEEE Wireless Commun., vol. 21, no. 2, pp. 106–112, Apr. 2014.
[2] A. K. Gupta, X. Zhang, and J. G. Andrews, “SINR and throughput scaling in ultradense urban cellular networks,” IEEE Wireless Commun. Lett., vol. 4, no. 6, pp. 605–608, Dec. 2015.
[3] T. Wu, W. Liao, and C.-J. Chang, “A cost-effective strategy for road-side unit placement in vehicular networks,” IEEE Trans. Commun., vol. 60, no. 8, pp. 2295–2303, Aug. 2012.
[4] X. You, H. Yin, and H. Wu, “On G6 and wide-area IoT,” Chin. J. Internet Things, vol. 4, no. 1, pp. 3–11, 2020.
[5] A. Merwaday and I. Guvenc, “UAV assisted heterogeneous networks for public safety communications,” in Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW), Mar. 2015, pp. 329–334.
[6] H. Wang, X. Zhou, and M. C. Reed, “Coverage and throughput analysis with a non-uniform small cell deployment,” IEEE Trans. Wireless Commun., vol. 13, no. 4, pp. 2047–2059, Apr. 2014.
[7] D. H. Kang, K. W. Sung, and J. Zander, “Attainable user throughput by dense Wi-Fi deployment at 5 GHz,” in Proc. IEEE PIMRC, Sep. 2013, pp. 3418–3422.
[8] J. Lyu and R. Zhang, “Spatial throughput characterization for intelligent reflecting surface aided multi-user system,” IEEE Wireless Commun. Lett., vol. 9, no. 6, pp. 834–838, Jun. 2020.
[9] H. Wang, J. Chen, G. Ding, and J. Sun, “Trajectory planning in UAV communication with jamming,” in Proc. 10th Int. Conf. Wireless Commun. Signal Process. (WCSP), Oct. 2018, pp. 1–6.
[10] S.-F. Chou, A.-C. Pang, and Y.-J. Yu, “Energy-aware 3D unmanned aerial vehicle deployment for network throughput optimization,” IEEE Trans. Wireless Commun., vol. 19, no. 1, pp. 563–578, Jan. 2020.
[11] W. He, H. Dai, and P. Ning, “Dynamic adaptive anti-jamming via controlled mobility,” IEEE Trans. Wireless Commun., vol. 13, no. 8, pp. 4374–4388, Aug. 2014.
[12] A. Rahmati et al., “Dynamic interference management for UAV-assisted wireless networks,” IEEE Trans. Wireless Commun., vol. 21, no. 4, pp. 2637–2653, Apr. 2022.
[13] P. Yang, X. Cao, X. Xi, Z. Xiao, and D. Wu, “Three-dimensional drone-cell deployment for congestion mitigation in cellular networks,” IEEE Trans. Veh. Technol., vol. 67, no. 10, pp. 9867–9881, Oct. 2018.
[14] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, “Efficient deployment of multiple unmanned aerial vehicles for optimal wireless coverage,” IEEE Commun. Lett., vol. 20, no. 8, pp. 1647–1650, Aug. 2016.
[15] M. Chen, M. Mozaffari, W. Saad, C. Yin, M. Debbah, and C. S. Hong, “Caching in the sky: Proactive deployment of cache-enabled unmanned vehicles for optimized quality-of-experience,” IEEE J. Sel. Areas Commun., vol. 35, no. 5, pp. 1040–1061, May 2017.
[16] D. Wang, M. Aziz, M. Helouei, and F. M. Ghanouchi, “Augmented real-valued time-delay neural network for compensation of distortions and impairments in wireless transmitters,” IEEE Trans. Neural Netw. Learn. Syst., vol. 30, no. 1, pp. 242–254, Jan. 2019.
[17] X. Hu et al., “Convolutional neural network for behavioral modeling and predistortion of wideband power amplifiers,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 8, pp. 3923–3937, Aug. 2022.
[18] S. Cai and V. K. N. Lau, “Remote state estimation of nonlinear systems over fading channels via recurrent neural networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 8, pp. 3908–3922, Aug. 2022.
[19] T. Ding and A. Hirose, “Fading channel prediction based on combination of complex-valued neural networks and chirp-Z-transform,” IEEE Trans. Neural Netw. Learn. Syst., vol. 25, no. 9, pp. 1686–1695, Sep. 2014.
[20] H. Su, H. Zhang, H. Jiang, and Y. Wen, “Decentralized event-triggered adaptive control of discrete-time nonzero-sum games over wireless sensor-actuator networks with input constraints,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 10, pp. 4254–4266, Oct. 2020.
[21] S. Abadal, A. Jain, R. Guirado, J. López-Alonso, and E. Alarcón, “Computing graph neural networks: A survey from algorithms to accelerators,” ACM Comput. Surv., vol. 54, no. 9, pp. 1–38, Dec. 2022.
[22] X. Zhang, H. Zhao, J. Xiong, X. Liu, Z. Zhou, and J. Wei, “Scalable power control/beamforming in heterogeneous wireless networks with graph neural networks,” in Proc. IEEE Global Commun. Conf., Dec. 2021, pp. 1–6.
[23] M. Eisen and A. Ribeiro, “Optimal wireless resource allocation with random edge graph neural networks,” IEEE Trans. Signal Process., vol. 68, pp. 2977–2991, 2020.
[24] J. Guo and C. Yang, “Learning power control for cellular systems with heterogeneous graph neural network,” in Proc. IEEE Wireless Commun. Netw. Conf. (WCNC), Mar. 2021, pp. 1–5.
[25] R. Liao, Z. Zhao, R. Urtasun, and R. Zemel, “LanczosNet: Multi-scale deep graph convolutional networks,” in Proc. Int. Conf. Learn. Represent, 2019, pp. 1–18.
[26] X. Jiang, P. Ji, and S. Li, “CensNet: Convolution with edge-node switching in graph neural networks,” in Proc. IJCAI, Aug. 2019, pp. 2656–2662.
[27] C. Morris et al., “Weisfeiler and Leman go neural: Higher-order graph neural networks,” in Proc. AAAI Conf. Artif. Intell., 2019, vol. 33, no. 1, pp. 4602–4609.
[28] W. Hu et al., “Strategies for pre-training graph neural networks,” in Proc. Int. Conf. Learn. Represent., 2020, pp. 1–22.
[29] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in Proc. Int. Conf. Learn. Represent., 2016, pp. 1–14.
[30] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 1263–1272.
[31] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 32, no. 1, pp. 4–24, Jan. 2021.
[32] O. Orhan, V. N. Swamy, T. Tetzlaff, M. Nassar, H. Nikopour, and S. Talwar, “Connection management in RAPP for O-RAN RIC: A graph neural network and reinforcement learning approach,” in Proc. 20th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA), Dec. 2021, pp. 936–941.
[33] S. Moon, S. Ahn, K. Son, J. Park, and Y. Yi, “Neuro-DFC: Design of wireless MAC via multi-agent reinforcement learning approach,” in Proc. ACM Mobihoc, Jul. 2021, pp. 141–150.
[34] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, “Spectral networks and locally connected networks on graphs,” in Proc. Int. Conf. Learn. Represent, 2020, pp. 1–14.
[35] R. Kondor, T. S. Hy, H. Pan, B. M. Anderson, and S. Trivedi, “Covariant compositional networks for learning graphs,” in Proc. Int. Conf. Learn. Represent. Workshop, 2018, pp. 1–19.
[36] D. Zou and G. Lerman, “Graph convolutional neural networks via scattering,” Appl. Comput. Harmon. Anal., vol. 49, no. 3, pp. 1046–1074, Nov. 2020.
[37] N. Keriven and G. Peyré, “Universal invariant and equivariant graph networks,” 2018, arXiv:1812.09902.
[38] R. Kondor, T. S. Hy, H. Pan, B. M. Anderson, and S. Trivedi, “Covariant compositional networks for learning graphs,” in Proc. Int. Conf. Learn. Represent. Workshop, 2018, pp. 1–19.
[39] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” Neural Netw., vol. 4, no. 2, pp. 251–257, 1991.
K. Xu, W. Hu, J. Leskovec, and S. Jegelka, “How powerful are graph neural networks?” in Proc. Int. Conf. Learn. Represent., 2019, pp. 1–17.

C. Bodnar et al., “Weisfeiler and Lehman go topological: Message passing simplicial networks,” in Proc. Int. Conf. Mach. Learn., 2021, pp. 1026–1037.

H. Maron, E. Fetaya, N. Segol, and Y. Lipman, “On the universality of invariant networks,” in Proc. Int. Conf. Learn. Represent., 2019, pp. 4363–4371.

T. Nguyen-Thien and T. Tran-Cong, “Approximation of functions and their derivatives: A neural network implementation with applications,” Appl. Math. Model., vol. 23, no. 9, pp. 687–704, Sep. 1999.

W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz, and R. Pascanu, “Sobolev training for neural networks,” in Proc. NeurIPS, 2017, pp. 4281–4290.

N. N. Vlassis and W. Sun, “Sobolev training of thermodynamic-informed neural networks for interpretable elasto-plasticity models with level set hardening,” Comput. Methods Appl. Mech. Eng., vol. 377, Apr. 2021, Art. no. 113695.

H. Son, J. W. Jang, W. J. Han, and H. J. Hwang, “Sobolev training for physics informed neural networks,” 2021, arXiv:2101.08932.

L. R. Ford and D. R. Fulkerson, “Maximal flow through a network,” Can. J. Math., vol. 8, pp. 399–404, 1956.

H. Kim, G. Papamakarios, and A. Mnih, “The Lipschitz constant of self-attention,” in Proc. Int. Conf. Mach. Learn., 2021, pp. 5562–5571.

H. Federer, Geometric Measure Theory, Berlin, Germany: Springer, 2014.

K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” Neural Netw., vol. 2, no. 5, pp. 359–366, Jan. 1989.

S. C. Brenner, L. R. Scott, and L. R. Scott, The Mathematical Theory of Finite Element Methods, vol. 3. New York, NY, USA: Springer, 2008.

J. He, L. Li, J. Xu, and C. Zheng, “ReLU deep neural networks and linear finite elements,” J. Comput. Math., vol. 38, no. 3, pp. 502–527, Jun. 2020.

W. Azizian and M. Lelarge, “Expressive power of invariant and equivariant graph neural networks,” in Proc. Int. Conf. Learn. Represent., 2021, pp. 1–39.

S. Jegelka, “Theory of graph neural networks: Representation and learning,” 2022, arXiv:2204.07697.

D. Hendrycks and K. Gimpel, “Gaussian error linear units (GELUs),” 2020, arXiv:2003.00982.

J. Sherman, “Breaking the multicommodity flow barrier for $O(\sqrt{n})$-approximations to sparsest cut,” in Proc. 50th Annu. IEEE Symp. Found. Comput. Sci., Oct. 2009, pp. 363–372.

S. Bhattacharya and T. Başar, “Graph-theoretic approach for connectivity maintenance in mobile networks in the presence of a jammer,” in Proc. IEEE CDC, Dec. 2010, pp. 3560–3565.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017, arXiv:1707.06347.

N. Heess et al., “Emergence of locomotion behaviours in rich environments,” 2017, arXiv:1707.02286.

I. Valiulahi and C. Masouros, “Multi-UAV deployment for throughput maximization in the presence of co-channel interference,” IEEE Internet Things J., vol. 8, no. 5, pp. 3605–3618, Mar. 2021.

D. K. Goldenberg, J. Lin, A. S. Morse, B. E. Rosen, and Y. R. Yang, “Towards mobility as a network control primitive,” in Proc. ACM MobiCom, May 2004, pp. 163–174.

M. Fey and J. E. Lenssen, “Fast graph representation learning with PyTorch geometric,” in Proc. ICLR Workshop Represent. Learn. Graphs Manifolds, 2019, pp. 1–9.

Yifei Yang received the B.S. degree in electronic information science and technology from Lanzhou University, Lanzhou, China, in 2018, and the M.S. degree in electromagnetic field and microwave technology from Wuhan University, Wuhan, China, in 2021, where he is currently pursuing the Ph.D. degree with the School of Electronic Information.

He joined Duke Kunshan University, Suzhou, China, in 2021, where he is currently a Research Intern in data science. His research interests include wireless networks, machine learning, and graph neural networks.

Dongmian Zou (Member, IEEE) received the B.S. degree (Hons.) in mathematics from The Chinese University of Hong Kong, Hong Kong, in 2012, and the Ph.D. degree in applied mathematics and scientific computation from the University of Maryland, College Park, MD, USA, in 2017.

After graduation, he served as a Post-Doctorate Researcher at the University of Minnesota at Twin Cities, Minneapolis, MN, USA. He joined Duke Kunshan University, Suzhou, China, in 2020, where he is currently an Assistant Professor of data science at the Division of Natural and Applied Sciences. He is also affiliated with the Zu Chongzhi Center for Mathematics and Computational Sciences and the Data Science Research Center. His research interests include intersection of applied harmonic analysis, machine learning, and signal processing.

Xiaofan He (Senior Member, IEEE) received the B.S. degree in electronics and information engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2008, the M.S. degree in electrical and computer engineering from McMaster University, Hamilton, ON, Canada, in 2011, and the Ph.D. degree in electrical and computer engineering from North Carolina State University, Raleigh, NC, USA, in 2015.

He was a tenure-track Assistant Professor of electrical engineering with Lamar University, Beaumont, TX, USA, from 2016 to 2018. He moved back to his hometown Wuhan in 2019 and is currently a Faculty Member with the Electronic Information School, Wuhan University, Wuhan. His research interests are in the general areas of wireless communications and networking. His current research mainly focuses on (distributed) edge computing and the associated optimization, scheduling, learning, and statistical analysis.

Dr. He received the Exemplary Reviewer Award of IEEE TRANSACTIONS ON COMMUNICATIONS in 2014 and 2015 and the Distinguished Member of Technical Program Committee Award of IEEE INFOCOM in 2018. He is currently serving as an Associate Editor for IEEE ACCESS.