A GNN Approach for Cell-Free Massive MIMO

Lou Salaün*, Hong Yang†, Shashwat Mishra* and Chung Shue Chen*

*Nokia Bell Labs, 1 Route de Villejust, Nozay, 91620 France,
†Nokia Bell Labs, 600 Mountain Avenue, Murray Hill, NJ 07974 USA,
Email: lou.salaun@nokia-bell-labs.com, h.yang@nokia-bell-labs.com,
shashwat.mishra@nokia.com, chung_shue.chen@nokia-bell-labs.com

Abstract—**Beyond 5G wireless technology Cell-Free Massive MIMO (CFmMIMO) downlink relies on carefully designed precoders and power control to attain uniformly high rate coverage. Many such power control problems can be calculated via second order cone programming (SOCP). In practice, several order of magnitude faster numerical procedure is required because power control has to be rapidly updated to adapt to changing channel conditions. We propose a Graph Neural Network (GNN) based solution to replace SOCP. Specifically, we develop a GNN to obtain downlink max-min power control for a CFmMIMO with maximum ratio transmission (MRT) beamforming. We construct a graph representation of the problem that properly captures the dominant dependence relationship between access points (APs) and user equipments (UEs). We exploit a symmetry property, called permutation equivariance, to attain training simplicity and efficiency. Simulation results show the superiority of our approach in terms of computational complexity, scalability and generalizability for different system sizes and deployment scenarios.**

Index Terms—**Cell-Free Massive MIMO, max-min power control, graph neural network, MRT, conjugate beamforming.**

I. INTRODUCTION

Since the seminal paper [1], Massive MIMO (mMIMO) now forms a backbone of 5G physical layer technology. With magnitudes more service antennas at the base stations, high precision beamforming becomes possible and uniform service quality can be made available in a cellular network for the first time [2], and the spectral efficiency has been greatly increased over 4G. Similar to previous generations of wireless technologies, to go beyond 5G, substantial increase in spectral efficiency is expected for future mobile networks.

To materially increase the spectral efficiency beyond cellular mMIMO, a cell-free version of mMIMO was first proposed in [3], and more detailed theoretical investigations were carried out in [4], [5]. Some key challenges in realizing CFmMIMO were outlined in [6]. Among them is the practical realization of downlink power control. Machine learning approaches to speed up the calculation of power controls are mostly motivated by real-world deployment requirements. Among existing literature, [7] used deep learning to perform uplink power allocation for sum-rate and max-min rate optimization; [8], [9] proposed unsupervised deep learning to obtain uplink power control for several optimization objectives. Downlink power control is in general different from the uplink as each access point (AP) is not only subject to total power constraint, but also must judiciously allocate powers to all served user equipments (UEs) for optimal fairness and interference mitigation. [10] proposed

a deep learning method to approximate a high complexity heuristic algorithm for max-min power control. [11], [12], [13], [14] considered neural network approach for the downlink power optimization. In particular, [13] employed a training data augmentation scheme and developed a CNN (convolutional neural network) to obtain a near optimal downlink max-min power control with reasonable complexity; [14] employed a deep reinforcement learning approach with deep deterministic policy gradient algorithm for the problem. It should be noted that the fully connected layers in [14] can incur large training complexity and thus limit the scalability and generalizability.

GNN [15] is a relatively recent neural network architecture that has received a lot of attention in the past few years. It associates graph components with sets of features that can be learned through iterative local computations. It can be very effective for node level, edge level, and graph level prediction tasks. In this paper, we develop a GNN to solve the downlink max-min power control problem with MRT precoding. Our contributions are:

1) We formulate the power control problem as a node level prediction task and design a graph structure that properly captures the dominant dependence relationship between APs and UEs. In our CNN approach [13], a symmetry property of CFmMIMO was utilized to vastly augment the training examples. This augmentation becomes unnecessary now as we recognize some intrinsic properties of the GNN, known as *permutation equivariance* [16]. It allows us to construct a GNN that matches the symmetric structure of CFmMIMO, which not only greatly simplifies the training process, but also achieves unprecedented accuracy, scalability, and reusability for different system sizes and deployment scenarios.

2) Our model optimizes directly each user's SINR by back-propagating the gradient of the SINR loss through the GNN during training. This further enables our GNN to produce near-optimal power control directly. In comparison, we used an additional convex optimization in our CNN approach [13].

3) We show through numerical simulations that a single trained GNN achieves practically near-optimal performance on a wide range of scenarios, with up to 128 APs, 32 UEs and various deployment morphologies. We compare the complexity of SOCP, CNN and GNN to confirm the advantage of GNN in large systems.

II. SYSTEM MODEL

A. Cell-Free Massive MIMO

We consider a CFmMIMO system [4], [5] with M APs deployed throughout the coverage area. All APs are connected to a central processing unit (CPU) for precoding and decoding processing and for power control coordination. We assume that each AP has one service antenna, and the M-AP system serves K single-antenna UEs simultaneously, where M is much larger than K.

The channel matrix between the M AP antennas and the K user antennas is denoted as:

$$\mathbf{G} = (\mathbf{g}_1 \quad \cdots \quad \mathbf{g}_K) = \begin{pmatrix} \bar{\mathbf{g}}_1^{\mathrm{T}} \\ \vdots \\ \bar{\mathbf{g}}_M^{\mathrm{T}} \end{pmatrix} \in \mathbb{C}^{M \times K},$$

where $\mathbf{g}_k \in \mathbb{C}^M$ is the channel vector between the k-th user and the M AP antennas, and $\bar{\mathbf{g}}_m \in \mathbb{C}^K$ is the channel vector between the m-th AP antenna and the K users, where the superscript $^{\mathrm{T}}$ denotes the transpose. The downlink data channel is modeled as:

$$\mathbf{x} = \mathbf{G}^{\mathrm{T}}(\sqrt{\rho_{\mathrm{d}}}\mathbf{s}) + \mathbf{w}, \tag{1}$$

where $\mathbf{x} \in \mathbb{C}^K$ is the received signal vector at the K user terminals, ρ_{d} is the downlink signal to noise ratio (SNR) for each AP, and $\mathbf{s} \in \mathbb{C}^M$ is M precoded inputs to the M antenna ports at the M APs, and $\mathbf{w} \in \mathbb{C}^K$ is a circularly-symmetric Gaussian noise vector. Each AP has a downlink power constraint, which can be specified as:

$$\|\mathbb{E}(\mathbf{s}^{*\mathrm{T}} \odot \mathbf{s})\|_{\infty} \le 1. \tag{2}$$

Here, $\mathbb{E}(\cdot)$ is the expectation, the superscript * denotes the complex conjugate transpose and $\odot$ denotes the element-wise multiplication.

B. Downlink Max-Min SINR with MRT

The channel between the m-th service antenna and the k-th user is denoted by:

$$g_{m,k} = \sqrt{\beta_{m,k}} h_{m,k}, \tag{3}$$

where $\beta_{m,k}$ models the large-scale fading that accounts for geometric attenuation and shadow fading, and $h_{m,k}$ models the small-scale fading that accounts for random scattering. In a rich scattering propagation environment, the magnitude of the signal typically varies randomly according to the Rayleigh distribution, thus the small-scaling fading $h_{m,k}$ are modeled as circularly symmetric complex Gaussian, independent and identically distributed random variables.

Under these assumptions, and with MMSE (minimum mean square error) channel estimation based on orthogonal uplink pilot sequence, the ergodic downlink effective SINR for the CFmMIMO system with MRT precoding is given by [5]:

$$\mathrm{SINR}_k = \frac{\rho_{\mathrm{d}} \left(\sum_{m=1}^M \sqrt{\alpha_{m,k}\eta_{m,k}}\right)^2}{1 + \rho_{\mathrm{d}} \sum_{m=1}^M \beta_{m,k} \sum_{k'=1}^K \eta_{m,k'}}, \tag{4}$$

where

$$\alpha_{m,k} = \frac{\rho_{\mathrm{u}}\tau\beta_{m,k}^2}{1 + \rho_{\mathrm{u}}\tau\beta_{m,k}}. \tag{5}$$

$\alpha_{m,k}$ is the mean-square of the channel estimate, and ρ_{d} and ρ_{u} are the normalized downlink and uplink SNR respectively. τ is the length of the uplink pilot sequence that is used for channel estimation. $\boldsymbol{\eta} = \{\eta_{m,k}\}$ is the downlink power control which is subject to per AP power constraint:

$$\sum_{k'=1}^K \eta_{m,k'} \le 1, \quad \forall m. \tag{6}$$

The max-min power control optimization problem can be formulated as

$$\max_{\boldsymbol{\eta}} \quad \min_k \quad \mathrm{SINR}_k,$$

$$\text{subject to} \quad \sum_{k'=1}^K \eta_{m,k'} \le 1, \quad \forall m, \tag{$\mathcal{P}$}$$

$$\eta_{m,k'} \ge 0, \quad \forall m, k'.$$

An optimal solution to problem $\mathcal{P}$ can be obtained with SOCP feasibility bisection search [4], [5], [17]. However, SOCP's computational complexity becomes impractical as M and K increase. For this reason, we develop a GNN that approximates the solution of SOCP but with practical run-times. The GNN takes as input the large scale fading coefficients, denoted by $\mathbf{B} = \{\beta_{m,k}\}$, and outputs the power control values $\boldsymbol{\eta} = \{\eta_{m,k}\}$. We will show that a single GNN model can achieve near-optimal performance over a wide range of system sizes and deployment morphologies.

III. GRAPH NEURAL NETWORK-BASED POWER CONTROL

A. Heterogeneous Graph Representation

We generate datasets composed of optimal $(\mathbf{B}, \boldsymbol{\eta})$ pairs, which are used to train and evaluate the neural network. Each dataset corresponds to a different simulation scenario characterized by a triplet (M, K, mor), where $mor \in \{urban, suburban, rural\}$ denotes the deployment and radio propagation morphology. For each example of the dataset, we generate $\mathbf{B}$ with $M \times K$ coefficients, as described in [17] and following the ITU-R specifications [18]. The corresponding power control $\boldsymbol{\eta}$ is obtained by SOCP.

One can notice that, if the AP indices $\{1, \ldots, M\}$ are permuted and/or the UE indices $\{1, \ldots, K\}$ are permuted at the input $\mathbf{B}$, then the optimal output would be $\boldsymbol{\eta}$ permuted in the same order. In other words, problem $\mathcal{P}$ does not depend on the choice of indices for the UEs and APs. This property is called *permutation equivariance*, and GNNs are known to inherently satisfy permutation invariance and equivariance [16].

To apply the GNN on our data, we need to represent them as graphs. We convert each data sample to a directed graph $\mathcal{G} = (V, E)$, where V is the set of nodes and E the set of directed edges. We create one node for each pair $(m, k) \in \{1, \ldots, M\} \times \{1, \ldots, K\}$, for a total of MK nodes. The one-to-one mapping between the (m, k) pair and the node index $i \in V = \{1, \cdots, MK\}$ is denoted by $\pi(m, k) = i$. Conversely,

we have $\pi^{-1}(i) = (m, k)$. Each node $i \in V$ is associated with a tensor h_i called *node feature*. The initial node features are the input **B** of the problem: for all $i \in V$, $h_i(0) = \beta_{m,k}$, where $\pi(m, k) = i$. Then, the GNN updates the node features through T iterations, i.e., $h_i(t)$, for $t = 1, \ldots, T$. The goal is to approximate the optimal output of the problem $\boldsymbol{\eta}$ with the final features: for all $i \in V$, $h_i(T) \approx \eta_{m,k}$, where $\pi(m, k) = i$. The intermediate features, for $t = 1, \ldots, T-1$, are called *hidden features* or *hidden layers*.

Since the GNN performs local computations on each node and their neighbors connected by an edge, we choose to create one edge between two nodes if they share the same AP or the same UE. There is no self-loop, i.e., $\forall i \in V$, $(i, i) \notin E$. Let $e \in E$, we define $\text{type}(e) \in \{\text{AP}, \text{UE}\}$ to be the edge type. Each edge is either of type AP if they share the same AP, or of type UE if they share the same UE. Hence, the edge construction is summarized by the following two statements, for all $m, m' \in \{1, \ldots, M\}$, $m \neq m'$, and $k, k' \in \{1, \ldots, K\}$, $k \neq k'$:

1) $e = (\pi(m, k), \pi(m, k')) \in E$ and $\text{type}(e) = \text{AP}$,
2) $e = (\pi(m, k), \pi(m', k)) \in E$ and $\text{type}(e) = \text{UE}$.

We consider heterogeneous edge types so the GNN can distinguish between these relationships and apply different operations on each type of edges. Let $i \in V$, we define its set of neighbors as:

$$\mathcal{N}(i) = \{j \in V \colon (i, j) \in E\}.$$

Similarly, we define the set of neighbors of type UE and type AP as:

$$\mathcal{N}_{\text{UE}}(i) = \{j \in V \colon (i, j) \in E \text{ and } \text{type}(i, j) = \text{UE}\},$$
$$\mathcal{N}_{\text{AP}}(i) = \{j \in V \colon (i, j) \in E \text{ and } \text{type}(i, j) = \text{AP}\}.$$

Fig. 1 depicts the neighbors of a typical node and its two types of edges. For $i \in V$, we have:

$$\mathcal{N}_{\text{UE}}(i) \cup \mathcal{N}_{\text{AP}}(i) = \mathcal{N}(i),$$
$$\mathcal{N}_{\text{UE}}(i) \cap \mathcal{N}_{\text{AP}}(i) = \emptyset.$$

Each $\mathcal{N}_{\text{UE}}(i)$ has $M - 1$ elements, each $\mathcal{N}_{\text{AP}}(i)$ has $K - 1$ elements, and each $\mathcal{N}(i)$ has $M + K - 2$ elements.

This node-edge structure associates nodes with 1st order dependence as neighbors and distinguishes the nodes that share APs and nodes that share UEs.

B. Data Preprocessing

The graph data preprocessing is done similarly to [13]. We first apply a $\log_2$ transformation to all the input and output features. This way, the features are within the same order of magnitude, and the GNN is able to extract useful information. As an example, if $\beta_{m,k}$ takes values in $\left[10^{-15}, 10^{-5}\right]$, then $-50 < \log_2 \beta_{m,k} < -16$. Then, we normalize the resulting values so that the input features and output features each have zero mean and unit standard deviation over all examples of the dataset. This is a common practice to help speed up the training, as the model does not have to learn the statistics of the data.
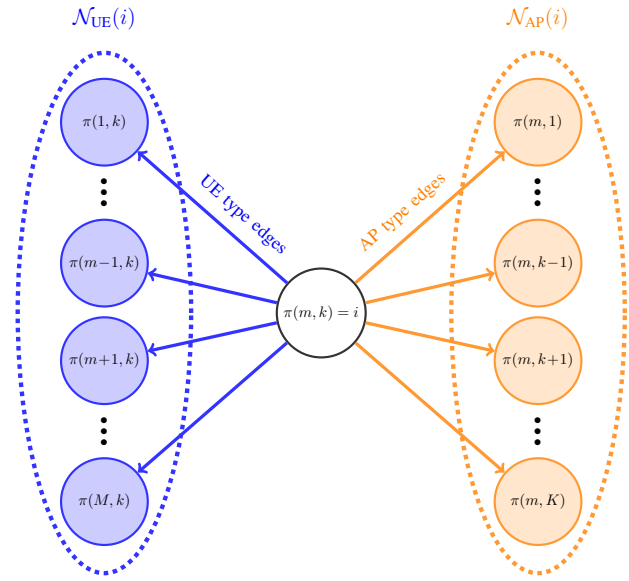

Fig. 1. A typical node $\pi(m, k) = i \in V$ and its two types of neighbors

C. The Structure of the Neural Network

In this sub-section, we use notation $\mathcal{L}$ to denote the linear operation that takes as input a tensor x of size n and outputs a tensor $\mathcal{L}(x)$ of size m as follows:

$$\mathcal{L}(x) = Wx + b,$$

where $W \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ are trainable parameters called *weight* and *bias*. Note that, the linear layers used in equations (8) and (9) do not share any trainable parameter. To differentiate them without ambiguity, we write $\mathcal{L}$ with different subscripts and superscripts.

The feature tensor of each node $i \in V$ is updated based on the features of its direct neighbors at the previous iteration. That is, $h_i(t + 1)$ is obtained from $h_i(t)$ and $h_j(t)$, for $j \in \mathcal{N}(i)$. The update rule is as follows:

$$h_i(t + 1) = \text{Norm}(\text{ReLU}(f_{\text{AP},t}(i) + f_{\text{UE},t}(i))), \quad (7)$$

where Norm denotes the layer normalization [19], and ReLU is the rectified linear unit activation function. The functions $f_{\bullet,t}$, for $\bullet \in \{\text{AP}, \text{UE}\}$, are inspired from the graph transformer of the UniMP model [20, Section 3.1], and are defined as:

$$f_{\bullet,t}(i) = \bigoplus_{c=1}^{C} \left(\mathcal{L}^1_{\bullet,c,t}(h_i(t)) \quad + \right.$$
$$\left. \sum_{j \in \mathcal{N}_\bullet(i)} \alpha_{\bullet,c,t}(i, j) \times \mathcal{L}^2_{\bullet,c,t}(h_j(t)) \right), \quad (8)$$

where $f_{\text{AP},t}(i)$ depends on the set of nodes $\mathcal{N}_{\text{AP}}(i) \cup \{i\}$, and $f_{\text{UE},t}(i)$ depends on $\mathcal{N}_{\text{UE}}(i) \cup \{i\}$. We implement each of these two functions with $C = 2$ attention heads. The two attention heads corresponding to $c = 1$ and $c = 2$ are concatenated into a single tensor. This concatenation operation is denoted by $\oplus$. Let d be the tensor size of each attention head, then $f_{\bullet,t}(i)$ is a tensor of size $2d$. Finally, $\alpha_{\bullet,c,t}(i, j)$ is the c-th attention

coefficient between source node i and destination node j, such that:

$$\alpha_{\bullet,c,t}(i,j) = \frac{\langle \mathcal{L}^3_{\bullet,c,t}(h_i(t)), \mathcal{L}^4_{\bullet,c,t}(h_j(t)) \rangle}{\sum_{u \in \mathcal{N}_{\bullet}(i)} \langle \mathcal{L}^3_{\bullet,c,t}(h_i(t)), \mathcal{L}^4_{\bullet,c,t}(h_u(t)) \rangle}, \quad (9)$$

where $\langle x,y \rangle = \exp\left(\frac{x^T y}{\sqrt{d}}\right)$, is the exponential scale dot-product [21] and d is the tensor size of each head.

The idea behind the above multi-head attention network is that each node can focus on a subset of its neighbors that are of interest instead of equally considering all the neighbors. Indeed, the attention coefficient controls which feature $h_j(t)$ will contribute to $h_i(t+1)$ through Eqn. (8): the feature is discarded if $\alpha_{\bullet,c,t}(i,j) \approx 0$, and kept otherwise. The level of dependence between two APs (or UEs) varies due to their relative geographic location. We use attention to efficiently learn the complex relationship between nodes. Furthermore, with multiple heads, different attention levels can be assigned to the various features of the same neighbor. Here, we implement $C = 2$ attention heads. A further increase in the number of heads does not significantly improve the performance for this problem. During our study, we observe that the attention mechanism greatly improves the GNN performance and generalizability to large number of APs and UEs.

We can see that whenever the input is permuted, the π mapping is permuted, but the neighborhood of each $\pi(m,k)$ node remains the same. Furthermore, the operations defined above do not depend on any ordering of the neighbors, since the neighboring features $h_j(t)$, for $j \in \mathcal{N}(i)$, are summed in (8). As a consequence, the GNN guarantees that the output is also permuted equivariantly. Thus, the GNN satisfies the permutation equivariance property of our problem $\mathcal{P}$.

By experiments, we optimized the structure of the GNN to achieve near-optimal performance with reasonable complexity. Our GNN model contains 9 hidden layers, i.e., $T = 10$, with the following node feature tensor sizes:

Layer sizes: $(\text{in} = 1, 8, 8, 16, 16, 32, 16, 16, 8, 8, \text{out} = 1)$.

As expected, the input and output both have a single value per node representing respectively the large scale fading coefficient, and the power control coefficient for each channel. Each hidden layer, for $t = 1, \ldots, 9$, is obtained from the previous layer by applying the multi-head attention neural network defined in Eqn. (7). The final output tensor is obtained by applying a simple linear activation of the form $h_i(T) = \mathcal{L}_{\text{out}}(h_i(T-1))$, for all node $i \in V$. To guarantee that the power constraints in $\mathcal{P}$ are satisfied, we apply the following two operations on the output tensor: (i) If any power coefficient is negative, we set it to zero. (ii) If the power budget constraint is violated for an AP m, then we renormalize its transmit powers. That is, if $\sum_{k'=1}^{K} \eta_{m,k'} > 1$, then for all UE k, we assign $\eta_{m,k} \leftarrow \eta_{m,k} / \sum_{k'=1}^{K} \eta_{m,k'}$.

D. Training and SINR Loss Function

The training dataset is composed of 4 scenarios: $(M, K, mor) = (32, 6, urban), (32, 9, urban), (64, 9, urban),$ $(64, 18, urban)$. Each scenario has 20,000 samples, for a total of 80,000 training samples. In comparison, the training dataset in [13] uses $36,000$ raw data for the $(32, 6, urban)$ and $(32, 9, urban)$ scenarios only. These data are then augmented to obtain 2.16×10^6 training samples. The augmentation consists of duplicating the samples 60 times and applying random row-permutations and column-permutations to them. This augmentation is needed for the CNN to learn the problem's permutation equivariance and achieve good performance. As explained in the previous sub-section, the GNN inherently satisfies this property. Therefore, such a data augmentation is not needed, and the GNN model converges faster during training. Furthermore, the CNN cannot be extended to larger scenarios without a massive amount of augmented data, as well as an increasing number of trainable parameters to represent the equivariance property. This is a major shortcoming of CNN, which we overcome with the proposed GNN.

The loss function used for training is the mean square error of the per-user SINR, which can be calculated as:

$$\frac{1}{K} \sum_{k=1}^{K} \left(\text{SINR}_k - \text{SINR}'_k \right)^2,$$

where SINR_k is the optimal SINR of user k obtained by SOCP, and SINR'_k is the SINR computed from the GNN predicted power coefficients $\boldsymbol{\eta}'$. The proposed loss function is differentiable and varies continuously with each user's SINR. By backpropagating the loss through the GNN, it adjusts its hidden layers based on which user's SINR should be increased or decreased at each training step.

We use the Adam optimizer [22] with a learning rate of 7×10^{-4} to train the GNN model. The batch size is 64, and the training is stopped after 100 epochs.

IV. Numerical Results

In this section, we show the performance of our GNN in terms of spectral efficiency, computational complexity, and generalizability for different system sizes and deployment morphologies. We compare it to the results obtained in [13] for the CNN and the optimal SOCP benchmark. To evaluate the complexity of each algorithm, we count their number of floating point operations (FLOPs) during execution. Each multiplication or addition counts as one FLOP.

The number of APs in our simulations ranges from $M = 5$ to 128. The number of UEs ranges from $K = 5$ to 32. These APs and UEs are randomly distributed in a circular area within a radius of 500 meters for the urban scenario, 1 km for suburban and 4 km for rural. We consider the "NLoS" propagation model specified in [18], and the path loss parameters used are the same as in [13].

The figures in this section show the spectral efficiency cumulative distribution function (CDF) achieved by the different algorithms for various simulation scenarios (M, K, mor). We generate 1,000 large-scale fading realizations for each simulation scenario. The *performance loss at median* refers to the relative difference in spectral efficiency between the

Scenario	FLOPs		Loss at median	
	GNN	CNN	GNN	CNN
Urban 24 APs, 5 UEs	1.5×10^7	3.7×10^6	0.72%	14.60%
Urban 32 APs, 5 UEs	1.9×10^7	3.7×10^6	0.48%	2.62%
Urban 32 APs, 9 UEs	3.2×10^7	3.7×10^6	0.58%	2.68%
Suburban 32 APs, 9 UEs	3.2×10^7	3.7×10^6	1.55%	2.81%
Rural 32 APs, 9 UEs	3.2×10^7	3.7×10^6	1.35%	2.84%

deep learning scheme (GNN or CNN) and the optimal solution obtained by SOCP, taken at the median of the CDF. The *95%-likely performance* refers to the spectral efficiency at the 5-th percentile, i.e., it indicates the coverage quality for 95% of the users.

All the simulations and algorithms are implemented in Python 3. The GNN is based on the PyTorch Geometric library [23]. The CNN is implemented in TensorFlow, and the SOCP problem $\mathcal{P}$ is solved using Mosek [24].

A. Comparison of GNN and CNN

We compare the performance of GNN and CNN on the scenarios simulated in paper [13]. The results are summarized in Table I. In the urban deployments with 32 APs, GNN achieves less than 0.6% performance loss at median, while CNN has about 2.6% loss. In the suburban and rural scenarios, the performance losses are respectively of 1.55% and 1.35% for the GNN versus 2.81% and 2.84% for the CNN. Besides, we see that the GNN generalizes well when the number of APs changes to 24, achieving similar performance loss of 0.72%, while the loss of the CNN degrades significantly to 14.6%.

In terms of FLOPs, GNN requires approximately 4 to 9 times more operations than CNN in these small systems. Nevertheless, it is implementable in practice as we observe an execution time of less than 100ms on a CPU[1]. We will see in the next subsection that the run-time of GNN remains practical when the number of APs and UEs increases.

B. GNN Spectral Efficiency and Complexity

Fig. 2 shows the spectral efficiency of GNN and SOCP on urban scenarios similar to the ones used for training. GNN reaches unprecedented accuracy, with less than 0.5% performance loss at median in all 4 scenarios. Moreover, the 95%-likely spectral efficiency is at most 0.017 bits/s/Hz away from optimal. This demonstrates the relevance of our graph representation for problem $\mathcal{P}$, and GNN's effectiveness in learning the optimal power control. Besides the figures, the simulation results of this subsection are summarized in Table II.

To see how our model generalizes to different graph sizes, we run simulations for $(M, K, mor) = (48, 12, urban)$, $(96, 30, urban)$ and $(128, 32, urban)$, see Fig. 3. Their performance loss at median are respectively 0.77%, 1.56% and
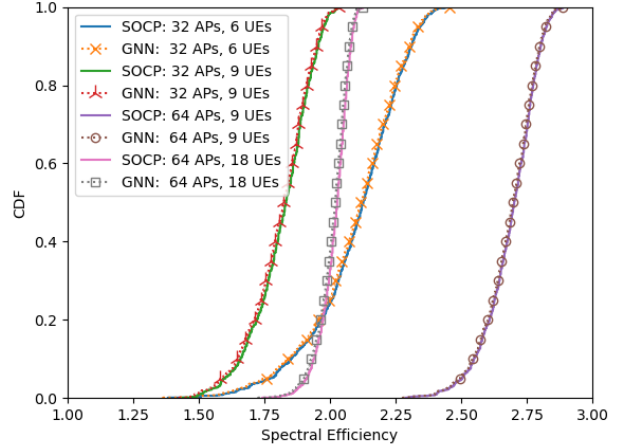

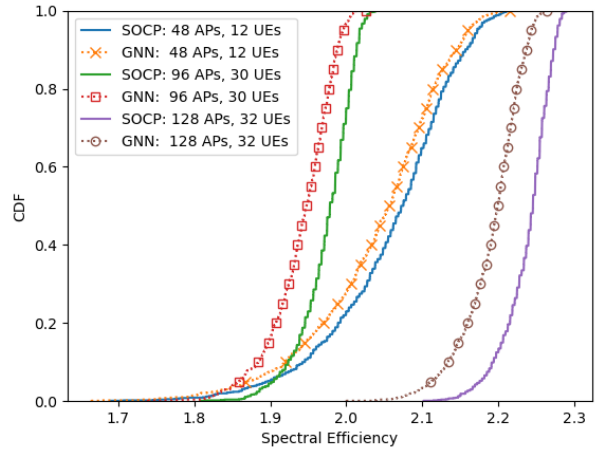Fig. 2. Spectral efficiency on urban scenarios

Fig. 3. Spectral efficiency on urban scenarios with graph sizes (number of UEs and APs) that have not been seen by the GNN during training

2.05%. We observe that the relative performance decreases slightly as the graph size increases. Overall, the performance remains very competitive even for graphs that are twice larger than the graphs used for training, e.g., 128 APs and 32 UEs.

We also validate our GNN model with other deployment morphologies. The suburban and rural results are presented in Table II. In brief, the performance loss at median is at most 1.54% for 32 and 64 APs, and at most 3.2% for 128 APs. The general trend in all these results is that the GNN inference can be slightly degraded by bigger graphs and different deployment morphologies than the ones used for training. However, the performance remains close to the optimal for all practical purposes.

Let us discuss about the computational complexity of our solution. The graph structure defined in Section III-A has MK nodes, $MK(M-1)$ edges of type UE and $MK(K-1)$ edges of type AP. Since the GNN operations are computed on the graph's nodes and edges, one can show that its asymptotic computational complexity is $O(MK(M+K))$. To validate this by simulation, we run the algorithms on randomly generated large-scale fading realizations for different values of $M = 1, \ldots, 256$ and $K = 5, \ldots, 72$, then we plot their FLOPs
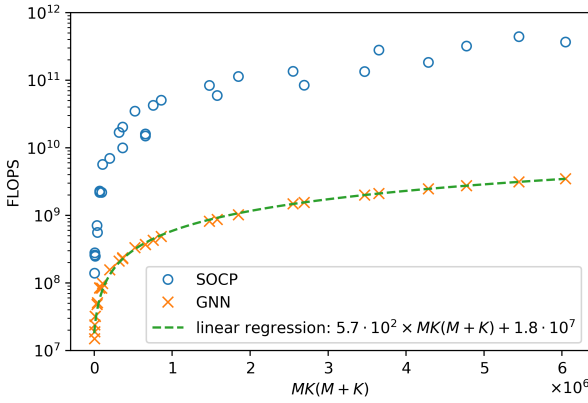
---

[1] with the following specifications: Intel Core i5 CPU with 8 GB of RAM, Windows 10, 64 bits.

Fig. 4. FLOPs of GNN and SOCP as a function of $MK(M + K)$

TABLE II
COMPARISON OF GNN AND SOCP

Scenario	FLOPs		GNN loss	GNN 95%
	GNN	SOCP	at median	likely loss
Urban:				
24 APs, 5 UEs	1.5×10^7	1.4×10^8	0.73%	1.42%
32 APs, 5 UEs	1.9×10^7	2.8×10^8	0.48%	1.19%
32 APs, 6 UEs	2.4×10^7	2.6×10^8	0.43%	0.51%
32 APs, 9 UEs	3.2×10^7	2.5×10^8	0.47%	1.05%
48 APs, 12 UEs	4.8×10^7	7.1×10^8	0.77%	1.51%
64 APs, 9 UEs	5.1×10^7	5.6×10^8	0.19%	0.25%
64 APs, 18 UEs	8.4×10^7	2.2×10^9	0.35%	0.57%
96 APs, 30 UEs	2.3×10^8	1.0×10^{10}	1.56%	2.39%
128 APs, 32 UEs	3.7×10^8	1.5×10^{10}	2.05%	2.94%
Suburban:				
32 APs, 9 UEs	3.2×10^7	2.5×10^8	1.54%	5.27%
64 APs, 18 UEs	8.5×10^7	2.2×10^9	1.20%	2.41%
128 APs, 32 UEs	3.7×10^8	1.6×10^{10}	3.20%	4.28%
Rural:				
32 APs, 9 UEs	3.2×10^7	2.5×10^8	1.35%	4.02%
64 APs, 18 UEs	8.4×10^7	2.3×10^9	0.78%	1.88%
128 APs, 32 UEs	3.7×10^8	1.6×10^{10}	2.97%	3.91%

versus $MK(M + K)$ in Fig. 4. We see that the GNN's FLOPs can be well fitted by a linear function in $MK(M + K)$ which confirms the above result.

In Fig. 4 and Table II, GNN has 10 times fewer FLOPs than SOCP on small scenarios with 32 APs. For 128 APs, it requires about 40 times fewer FLOPs than SOCP, and for 256 APs, it reduces the FLOPs by a factor 105. The same observation can be made on the run-times: on a CPU[1], the GNN computes the power control for 128 APs and 32 UEs in 1s, while it takes 45s for SOCP to complete. In real-world systems, the GNN would be implemented on a GPU: we observe under 50ms of run-times on a Nvidia TITAN RTX GPU.

V. CONCLUSION

In this paper, we propose a GNN to solve the downlink max-min power control for a CFmMIMO system with MRT precoded beamforming. Our solution takes advantage of the problem's permutation equivariance property to greatly improve the learning efficiency and accuracy. Numerical results show that a single trained GNN achieves near-optimal performance for various systems sizes and deployment scenarios.

Furthermore, its complexity remains low in all the simulated use-cases, therefore it is implementable in practice even in very large systems. The aforementioned two points demonstrate the superiority of our approach over the state of the art in terms of scalability and generalizability.

REFERENCES

[1] T. L. Marzetta, "Noncooperative cellular wireless with unlimited numbers of base station antennas," *IEEE Trans. Wireless Commun.*, vol. 9, no. 11, pp. 3590–3600, 2010.

[2] H. Yang and T. L. Marzetta, "A macro cellular wireless network with uniformly high user throughputs," in *IEEE Veh. Technol. Conf.*, 2014.

[3] ——, "Capacity performance of multicell large-scale antenna systems," in *51st Annual Allerton Conference on Communication, Control, and Computing*, 2013.

[4] H. Q. Ngo, A. Ashikhmin, H. Yang, E. G. Larsson, and T. L. Marzetta, "Cell-free massive MIMO versus small cells," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1834–1850, 2017.

[5] E. Nayebi, A. Ashikhmin, T. L. Marzetta, H. Yang, and B. D. Rao, "Precoding and power optimization in cell-free massive MIMO systems," *IEEE Trans. Wireless Commun.*, vol. 16, no. 7, pp. 4445–4459, 2017.

[6] J. Zhang, E. Björnson, M. Matthaiou, D. W. K. Ng, H. Yang, and D. J. Love, "Prospective multiple antenna technologies for beyond 5G," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 8, pp. 1637–1660, 2020.

[7] C. D'Andrea, A. Zappone, S. Buzzi, and M. Debbah, "Uplink power control in cell-free massive MIMO via deep learning," in *IEEE 8th International Workshop on Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP)*, 2019.

[8] Y. Zhang, J. Zhang, Y. Jin, S. Buzzi, and B. Ai, "Deep learning-based power control for uplink cell-free massive MIMO systems," in *IEEE Globecom*, 2021.

[9] N. Rajapaksha, K. B. S. Manosha, N. Rajatheva, and M. Latva-aho, "Deep learning-based power control for cell-free massive MIMO networks," in *IEEE International Conference on Communications*, 2021.

[10] Y. Zhao, I. G. Niemegeers, and S. Heemstra de Groot, "Power allocation in cell-free massive MIMO: A deep learning method," *IEEE Access*, vol. 8, no. 5, pp. 87 185–87 200, 2020.

[11] H. Yan, A. Ashikhmin, and H. Yang, "Optimally supporting IoT with cell-free massive MIMO," in *IEEE Globecom*, 2020.

[12] ——, "A scalable and energy efficient IoT system supported by cell-free massive MIMO," *IEEE Internet Things J.*, 2021.

[13] L. Salaün and H. Yang, "Deep learning based power control for cell-free massive MIMO with MRT," in *IEEE Globecom*, 2021.

[14] L. Luo, J. Zhang, S. Chen, X. Zhang, B. Ai, and D. W. K. Ng, "Downlink power control for cell-free massive MIMO with deep reinforcement learning," *IEEE Trans. Veh. Technol.*, 2022, early Access.

[15] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, 2009.

[16] N. Keriven and G. Peyré, "Universal invariant and equivariant graph neural networks," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[17] H. Yang and T. L. Marzetta, "Energy efficiency of massive MIMO: cell-free vs. cellular," in *IEEE 87th Veh. Technol. Conf.*, 2018.

[18] M. Series, "Guidelines for evaluation of radio interface technologies for IMT-Advanced," *Report ITU M.2135-1*, 2009.

[19] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[20] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," *arXiv preprint arXiv:2009.03509*, 2020.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[24] E. D. Andersen, C. Roos, and T. Terlaky, "On implementing a primal-dual interior-point method for conic quadratic optimization," *Mathematical Programming*, vol. 95, no. 2, pp. 249–277, 2003.