

# LORA: Learning to Optimize for Resource Allocation in Wireless Networks with Few Training Samples

Yifei Shen, *Student Member, IEEE*, Yuanming Shi, *Member, IEEE*, Jun Zhang, *Senior Member, IEEE*, and Khaled B. Letaief, *Fellow, IEEE*

## Abstract

Effective resource allocation plays a pivotal role for performance optimization in wireless networks. Unfortunately, typical resource allocation problems are mixed-integer nonlinear programming (MINLP) problems, which are NP-hard in general. Machine learning-based methods recently emerge as a disruptive way to obtain near-optimal performance for MINLP problems with affordable computational complexity. However, a key challenge is that these methods require huge amounts of training samples, which are difficult to obtain in practice. Furthermore, they suffer from severe performance deterioration when the network parameters change, which commonly happens and can be characterized as the *task mismatch* issue. In this paper, to address the sample complexity issue, instead of directly learning the input-output mapping of a particular resource allocation algorithm, we propose a *Learning to Optimize* framework for Resource Allocation, called *LORA*, that learns the pruning policy in the optimal branch-and-bound algorithm. By exploiting the algorithm structure, this framework enjoys an extremely low sample complexity, in the order of hundreds, compared with millions for existing methods. To further address the task mismatch issue, we propose a transfer learning method via self-imitation, named *LORA-TL*, which can adapt to the new task with only a few additional *unlabeled* training samples. Numerical simulations demonstrate that LORA outperforms specialized state-of-art algorithms and achieves near-optimal performance. Moreover, LORA-TL, relying on a few unlabeled samples, achieves comparable performance with the model trained from scratch with sufficient labeled samples.

The materials in this paper were presented in part at IEEE Global Conference on Signal and Information Processing (GlobalSIP), 2018 [1], and submitted to IEEE International Conference on Communications (ICC), 2019.

Y. Shen, J. Zhang, and K. B. Letaief are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong (E-mail: {yshenaw, eejzhang, eekhaled}@ust.hk).

Y. Shi is with the School of Information Science and Technology, ShanghaiTech University, Shanghai 201210, China (E-mail: shiym@shanghaitech.edu.cn).

## Index Terms

Resource allocation, mixed-integer programming, wireless networks, few-shot learning, imitation learning, transfer learning.

## I. INTRODUCTION

### *A. Motivations*

In wireless networks, effective allocation of radio resources is vital for performance optimization [2]. Unfortunately, typical resource allocation problems, such as subcarrier allocation in OFDMA [3], user association [4], access point selection [5], and computation offloading [6], are mixed integer nonlinear programming (MINLP) problems, which are NP-hard in general. The complexity of global optimization algorithms, e.g., the branch-and-bound algorithm, is exponential. Thus, most of the existing studies focused on sub-optimal or heuristic algorithms, whose performance gaps to the optimal solution are difficult to quantify and control.

Machine learning recently emerges as a disruptive technology to balance the computational complexity and the performance gap for solving NP-hard problems, and has attracted lots of attention from the mathematical optimization community [7]. This trend has also inspired researchers to apply machine learning-based methods to solve optimization problems in wireless networks. For example, for the interference channel power management problem, it was proposed in [8] to accelerate the classic weighted minimum mean square error (WMMSE) [9] algorithm by using the multilayer perceptron (MLP) to approximate the solution given by WMMSE. However, as the training samples are obtained via a sub-optimal algorithm, i.e., WMMSE, there is a performance gap compared with the optimal solution. To achieve near-optimal performance, unsupervised learning methods have been proposed in [10], [11], which do not depend on any existing resource allocation algorithm. Besides improving performance and computational efficiency, machine learning techniques have also been applied to deal with other issues in resource allocation. In particular, deep reinforcement learning [12], spatial deep learning [13], and deep neural network parameterization [14] have been proposed to deal with the scenarios with delayed channel state information (CSI), without CSI but only geographical locations, and unknown resource allocation functions, respectively.

While the above attempts demonstrated the great potential of the “learning to optimize” approach for resource allocation in wireless networks, applying them into real systems faces additional difficulties. A prominent shortcoming of these algorithms is that they require large amounts of training samples, e.g., millions of samples are needed for a small-size system [10]. The acquisition of such large amount of

samples is prohibitive and impractical. Secondly, wireless networks are inherently dynamic, e.g., both the locations and number of users are changing dynamically. Thus, the pre-trained machine learning model may be useless or suffer from severe performance deterioration as the network setting changes. This issue can be characterized as *task mismatch*, i.e., the test setting is different from the trained one. Finally, resource allocation problems are constrained by nature, but the ability of existing machine learning-based methods on dealing with constraints is limited [7]. To address these challenges, in this paper we shall develop a novel machine learning framework for MINLP resource allocation problems in wireless networks, which requires a few training samples, is able to effectively handle task mismatch, and achieves near-optimal performance with feasibility guarantee.

### B. Literature Review

TABLE I  
SUMMARY OF “LEARNING TO OPTIMIZE” METHODS.

	<b>End-to-end learning</b>	<b>Optimization policy learning</b>
Methodology <sup>1</sup>	$\underset{\Theta}{\text{minimize}} \quad d(\hat{a}, a^*, \Theta)$ minimize the distance between the output of learning algorithm and optimal solution	$\underset{\Theta}{\text{minimize}} \quad d(\hat{\pi}, \pi^*, \Theta)$ minimize the distance between the learned policy and optimal policy
Samples required	Large	Few
Constraint feasibility	By Lagrange multipliers and projection, with performance loss	By the optimization algorithm, without performance loss
Generalization ability	Weak	Strong
Outperform the labels <sup>2</sup>	No	Yes
Applications in machine learning	[15], [16], [17]	[18], [19], [20]
Applications in wireless networks	[8], [10], [11]	LORA, LORA-TL

This investigation sits at the intersection of wireless communications, mathematical optimization, and machine learning. To set the stage, we first review available results on “learning to optimize” in these different areas. The goal of “learning to optimize” is to obtain near-optimal algorithms with affordable

<sup>1</sup> $d(\cdot, \cdot)$  denotes some distance metric,  $\hat{a}$  and  $\hat{\pi}$  denote the learned solution and learned policy,  $a^*$  and  $\pi^*$  denote the optimal solution and optimal policy, and  $\Theta$  denotes the learning parameter.

<sup>2</sup>This comparison is for the “supervised learning” setting, and the labels may not be the optimal solutions, e.g., labeled by the sub-optimal WMMSE algorithm as in [8].

computational complexity for challenging optimization problems. This topic has been developed for decades in the machine learning community [21], and has also attracted significant attention from mathematical optimization and operation research [7]. Many approaches have been proposed under this framework, as listed in Table I.

The first paradigm is “end-to-end” learning, which outputs solutions directly from the input problem data. A straightforward idea is to regard the optimization algorithm as a black-box, and learns the input-output mapping [21]. For example, the differentiable neural computer [15] and pointer networks [16] have been proposed for approximating the output of the algorithms for combinatorial optimization problems. In wireless networks, the method proposed in [8] applied this idea. The adopted neural networks in these studies have generic architectures and perform well in practical problems. Nevertheless, they require large amounts of samples for training, which impedes using optimal solutions as labels when solving MINLP problems. Thus, the performance of the trained model is upper bounded by the adopted sub-optimal algorithms to generate labels. Besides, the constraint feasibility is guaranteed by simple projection, which will result in performance losses. In order to achieve near-optimal performance without optimal labels, the learning algorithm must be aware of the optimization problem to be solved. A commonly used approach is to treat the objective function of the optimization problem as the loss or reward function of the learning algorithm [22], [17]. In wireless networks, the studies in [10], [11], [12] followed this approach, and achieve near-optimal performance without labels. For constrained problems, to reduce the performance loss due to the projection, a new loss function can be defined to penalize the constraint violation [23], which is also used in [10] in wireless networks. However, these methods still require a huge amount of training samples.

Few-shot learning, i.e., learning from few training samples, is an important paradigm in machine learning [24], and it mimics the way humans learn new concepts [25]. Such an approach has the potential to significantly reduce the sample complexity of the above-mentioned studies. In particular, applying machine learning alongside specific optimization algorithms has been proposed [7], which is referred as “optimization policy learning” in Table I. This method can reduce the sample complexity as the prior knowledge is exploited, i.e., the structure of the optimization algorithm. Additionally, learning the policies in a specific optimization algorithm will have no influence on the feasibility guarantee of that algorithm, which thus can nicely handle constrained problems. Optimization policy learning has recently attracted lots of attention, especially for combinatorial optimization problems. For mixed-integer linear programming problems, it was proposed in [20] to learn whether to use a decomposition in the branch-and-cut algorithm and search policies in the branch-and-bound algorithm [18]. Heuristics are learned in the greedy algorithm for combinatorial optimization problems over graphs [19]. Besides few training samples and feasibility

guarantee, there are other advantages by exploiting the algorithm structure. Firstly, if the labels are not optimal, the learned policy can outperform the labels by exploration [7], [26]. Secondly, this kind of methods enjoy good generalization abilities, e.g., scale up to larger problem sizes and generalize to different problems [18], [19]. Inspired by its unique advantages mentioned above, our proposed approach is based on “optimization policy learning”.

### C. Contributions

In this paper, we focus on finding near-optimal solutions for MINLP resource allocation problems in wireless networks by machine learning with few training samples. A main innovation is a learning to optimize framework for resource allocation, called LORA. The merits of this framework and major contributions of this paper are summarized as follows:

- 1) We propose the LORA framework based on imitation learning. Specifically, we formulate the pruning policy in the branch-and-bound algorithm as a sequential decision problem, followed by learning the optimal pruning policy via imitation learning. This framework achieves near-optimal performance with few training samples by exploiting the structure of the branch-and-bound algorithm. The feasibility for constraints is guaranteed by iteratively increasing the search space. Furthermore, LORA can perform better than the labels and scale up to larger problem sizes.
- 2) To tackle the task mismatch issue, we propose a transfer learning method via self-imitation, which leads to the LORA-TL framework. Compared with the traditional transfer learning method, i.e., fine-tuning, LORA-TL only requires a few additional *unlabeled* training samples. It is based on the principle of transfer learning [27] and self-imitation learning [26]. Specifically, we first train a policy for a network setting where abundant labeled samples are readily available. The learned policy is then blended with an exploration policy to explore the branch-and-bound tree for additional training samples in the new scenario. The best solution found is served as the training labels of the additional training samples, followed by fine-tuning the learned policy with these labels.
- 3) We test LORA and LORA-TL on two applications, i.e., interference channel power control [28] and network power minimization [5]. Simulations demonstrate the effectiveness of LORA and LORA-TL in the following aspects:
  - a) LORA outperforms the specialized state-of-art methods and achieves near-optimal performance under a variety of system configurations, with only tens or hundreds of training samples.
  - b) It is demonstrated that, with LORA, a model trained on one setting can achieve state-of-art performance under moderately different settings, e.g., with different large-scale fading and numbers

of users or base stations. Such generalization capability is achieved via exploiting the algorithm structure and careful feature design.

- c) When the variation of the system configuration becomes dramatic, directly generalizing the model trained via LORA to the new setting induces considerable performance degradation. In this case, LORA-TL is able to achieve comparable performance with the model trained on sufficient samples for the test setting from scratch, and it only relies tens of additional unlabeled training samples.

## II. RESOURCE ALLOCATION AND BRANCH-AND-BOUND

In this section, we shall first introduce a general formulation for mixed-integer resource allocation problems in wireless networks. Then a global optimization algorithm, i.e., branch-and-bound, will be introduced, followed by some observations.

### A. Mixed-integer Resource Allocation

A wide range of resource allocation problems in wireless networks can be formulated as MINLP problems, which consist of a discrete optimization variable  $\mathbf{a}$ , e.g., indicating cell association or subcarrier allocation, and a continuous optimization variable  $\mathbf{w}$ , e.g., transmit power, subject to resource or performance constraints. Typical examples include subcarrier allocation in OFDMA [3], user association [4], network power minimization in cloud radio access networks (Cloud-RANs) [5], and joint task offloading scheduling and transmit power allocation [6]. A general formulation for such problems is given by

$$\begin{aligned} \mathcal{P} : & \underset{\mathbf{a}, \mathbf{w}}{\text{minimize}} && f(\mathbf{a}, \mathbf{w}) \\ & \text{subject to} && \mathcal{Q}(\mathbf{a}, \mathbf{w}) \leq 0 \\ & && a[i] \in \mathbb{N}, w[i] \in \mathbb{C}, \end{aligned} \tag{1}$$

where  $f(\cdot, \cdot)$  is an objective function, e.g., the sum rate or network power consumption,  $a[i]$  and  $w[i]$  are the elements of  $\mathbf{a}$  and  $\mathbf{w}$  respectively, and  $\mathcal{Q}(\cdot, \cdot)$  represents constraints such as the QoS or power constraint. MINLP problems are NP-hard in general. Optimal solutions can be found by the global optimization algorithms, one of which will be explored in this paper.

### B. Branch-and-Bound

The branch-and-bound algorithm is one of the state-of-art algorithms to find an optimal solution to the MINLP problems [29]. It consists of three main policies: the node selection policy, the variable selection policy, and the pruning policy, which together construct a binary search tree iteratively.

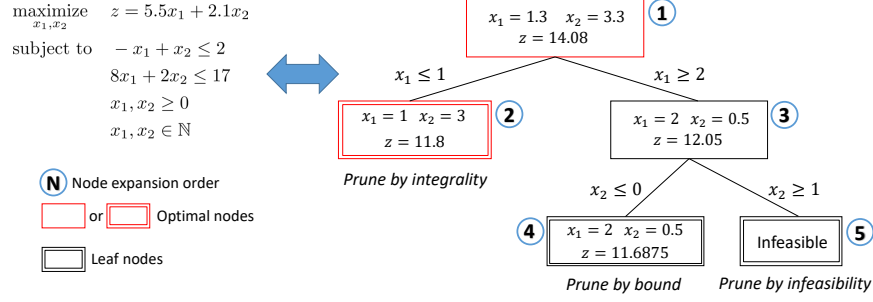


Fig. 1. An example of the branch-and-bound algorithm [29]. The solution of the relaxed linear programming problem is  $x_1 = 1.3$ , and  $x_2 = 3.3$ . It is used as the initial solution for the tree search. The objective value  $z = 14.08$  provides an upper bound to the original problem. According to the branching rule, we branch out to the first variable  $x_1$  and create two sub-problems. According to the node selection policy, we select the left child. The solution to the relaxed problem of the left child is  $x_1 = 1, x_2 = 3$ . Because they are integer values, we prune the node by integrality. The objective value  $z = 11.8$  provides a lower bound for the original problem. We then turn to the right child of the root node. We solve it and branch out on  $x_2$ , which generates two child nodes. The optimal objective value of the left child is  $11.6875 < 11.8$ , so we prune it by bound. The relaxed problem of the right child is infeasible and hence we prune it by infeasibility. Thus, the whole branch-and-bound tree is visited and we get the solution to the original problem  $x_1 = 1, x_2 = 3$ .

We first specify some notations. Each node in the binary search tree contains a MINLP problem and its relaxed problem. Let  $N_t$  denote the node selected at the  $t$ -th iteration,  $\mathcal{P}_t$  denote the MINLP problem at node  $N_t$ , and  $\mathcal{G}_t$  represent the feasible set of  $\mathcal{P}_t$ . The natural relaxation of  $\mathcal{G}_t$ , i.e., relax all the integer constraints into box constraints, is denoted as  $\mathcal{H}_t$ . For example, binary constraints  $\mathcal{G}_t = \{(\mathbf{a}, \mathbf{w}) : a[i] \in \{0, 1\}, \forall i\}$  are relaxed into box constraints  $\mathcal{H}_t = \{(\mathbf{a}, \mathbf{w}) : a[i] \in [0, 1], \forall i\}$ . Thus, the original problem and relaxed problems at node  $N_t$  can be represented respectively as  $\underset{\mathbf{a}, \mathbf{w}}{\text{minimize}} \{f(\mathbf{a}, \mathbf{w}) : (\mathbf{a}, \mathbf{w}) \in \mathcal{G}_t\}$  and  $\underset{\mathbf{a}, \mathbf{w}}{\text{minimize}} \{f(\mathbf{a}, \mathbf{w}) : (\mathbf{a}, \mathbf{w}) \in \mathcal{H}_t\}$ . Let  $(\mathbf{a}_t^*, \mathbf{w}_t^*)$  and  $z_t^*$  respectively denote the optimal solution and the optimal objective value of the relaxed problem at  $N_t$ , i.e.,

$$(\mathbf{a}_t^*, \mathbf{w}_t^*) = \arg \min_{\mathbf{a}, \mathbf{w}} \{f(\mathbf{a}, \mathbf{w}) : (\mathbf{a}, \mathbf{w}) \in \mathcal{H}_t\}$$

$$z_t^* = \min_{\mathbf{a}, \mathbf{w}} \{f(\mathbf{a}, \mathbf{w}) : (\mathbf{a}, \mathbf{w}) \in \mathcal{H}_t\}.$$
(2)

We use  $a_t^*[j]$  to represent  $j$ -th element of  $\mathbf{a}_t^*$ . The best integer solution to  $\mathcal{P}$  found before or in this iteration is denoted as  $c^*$ .

The branch-and-bound algorithm maintains an unexplored node list  $\mathcal{L}$ , and this list only contains the root node  $N_1$  at the beginning. The MINLP problem at the root node is  $\mathcal{P}$ . At each iteration, the node selection policy first selects a node and pops it from the unexplored list. The relaxed problem at this node is solved to obtain  $\mathbf{a}_t^*$  and  $z_t^*$ . Then the pruning policy  $\pi_p$  determines whether to preserve this node

according to  $\mathbf{a}_t^*$  and  $z_t^*$ . The algorithm enters the next iteration if the pruning policy decides to prune this node, i.e., no child node of this node will be considered. Otherwise, the variable selection policy chooses a variable index  $j$  such that  $a_t^*[j]$  is fractional. Then the feasible set  $\mathcal{G}_t$  is divided into two parts as

$$\begin{aligned}\mathcal{G}_t^1 &= \mathcal{G}_t \cap \{(\mathbf{a}, \mathbf{w}) : a[j] \leq \lfloor a_t^*[j] \rfloor\}, \\ \mathcal{G}_t^2 &= \mathcal{G}_t \cap \{(\mathbf{a}, \mathbf{w}) : a[j] \geq \lceil a_t^*[j] \rceil\}.\end{aligned}\tag{3}$$

Two new MINLP problems are formed based on the partition

$$\begin{aligned}\mathcal{P}_t^1 &: \underset{\mathbf{a}, \mathbf{w}}{\text{minimize}} \{f(\mathbf{a}, \mathbf{w}) : (\mathbf{a}, \mathbf{w}) \in \mathcal{G}_t^1\}, \\ \mathcal{P}_t^2 &: \underset{\mathbf{a}, \mathbf{w}}{\text{minimize}} \{f(\mathbf{a}, \mathbf{w}) : (\mathbf{a}, \mathbf{w}) \in \mathcal{G}_t^2\}.\end{aligned}\tag{4}$$

Two child nodes  $N_t^1$  and  $N_t^2$ , whose MINLP problems are  $\mathcal{P}_t^1$  and  $\mathcal{P}_t^2$  respectively, are produced and put into the unexplored node list  $\mathcal{L}$ . These procedures repeat at each iteration until there is no nodes in  $\mathcal{L}$ . The pseudo-code is shown in Algorithm 1 and an illustration is shown in Fig. 1.

---

**Algorithm 1** The branch-and-bound algorithm.

---

```

1:  $\mathcal{L} \leftarrow \{N_1\}, c^* \leftarrow +\infty, t \leftarrow 0$ 
2: while  $\mathcal{L} \neq \emptyset$  do
3:    $t \leftarrow t + 1$ 
4:    $N_t \leftarrow$  selects a node from  $\mathcal{L}$ 
5:    $\mathcal{L} \leftarrow \mathcal{L} \setminus \{N_t\}$ 
6:    $(\mathbf{a}_t^*, \mathbf{w}_t^*), z_t^* \leftarrow$  solve the relaxed problem of  $\mathcal{P}_t$ 
7:   if  $\pi_p(N_t, \mathbf{a}_t^*, z_t^*) = \text{preserved}$  then
8:      $N_t^1, N_t^2 \leftarrow$  two children of  $N_t$ 
9:      $\mathcal{L} \leftarrow \mathcal{L} \cup \{N_t^1, N_t^2\}$ 
10:  end if
11:  if  $a_t^*[j], \forall j$  is integer then
12:     $c^* \leftarrow \min(c^*, z_t^*)$ 
13:  end if
14: end while
```

---

In the standard branch-and-bound algorithm, the pruning policy  $\pi_p$  prunes a node only if one of the following conditions is met:



- *Prune by bound*:  $z_t^* \geq c^*$ .  $z_t^*$  provides a lower bound for the optimal objective value of  $\mathcal{P}_t$  since  $\mathcal{G}_t \subseteq \mathcal{H}_t$ . Under such circumstance, the optimal solution to  $\mathcal{P}_t$  cannot be the best solution as its lower bound is worse than the best objective value found.
- *Prune by infeasibility*: The relaxed problem of  $\mathcal{P}_t$  is infeasible. This is a special case of *prune by bound* if we viewed the infeasibility as  $z_t^* = +\infty$ .
- *Prune by integrality*:  $a_t^*[j], \forall j$ , is an integer. It is unnecessary to search the children of  $N_t$  since we have found the optimal solution to  $\mathcal{P}_t$ .

### C. Observations

Branch-and-bound has been widely employed in solving MINLPs in wireless networks, as it is capable to obtain the globally optimal solution. However, its computational complexity is exponential in the dimension of  $\mathbf{a}$ , which can not be tolerated in practice. Thus, lots of studies resort to heuristic algorithms [5], [9], which, however, usually have non-negligible performance gaps to the globally optimal solution. It is observed in the search procedure that the computational complexity is mainly determined by the pruning policy. The more nodes are pruned, the less time the branch-and-bound algorithm needs. The pruning policy in the standard branch-and-bound algorithm is conservative because the optimality must be guaranteed. The optimality is guaranteed by checking that all other solutions are worse than the returned solution. Therefore, most of the running time of the branch-and-bound algorithm is spent on checking the non-optimal nodes. In practical situations, what we want is a good solution rather than the optimality guarantee. Thus, we can dramatically boost the efficiency of the branch-and-bound algorithm by learning an aggressive pruning policy.

## III. LORA: LEARNING TO OPTIMIZE FOR RESOURCE ALLOCATION

In this section, the LORA framework is presented, we first give motivations for adopting imitation learning as the main method to develop the framework. The details of LORA are then introduced, including the implementation issues, design of the classifier and features.

### A. LORA via Imitation Learning

In this subsection, we first model the procedure of the branch-and-bound algorithm as a *sequential decision problem*, then discuss why we employ imitation learning to learn the policies of this problem. Key elements of the LORA framework are presented next.

A sequential decision problem consists of a state space  $\mathcal{X}$ , an action space  $\mathcal{Q}$ , and a policy space  $\Pi$ . A single trajectory  $\tau$  consists of a sequence of states  $x_1, \dots, x_T \in \mathcal{X}$ , a sequence of actions  $q_1, \dots, q_T \in \mathcal{Q}$ ,

and a policy  $\pi \in \Pi$  mapping an observation to an action  $\pi(x_t) = q_t$ . In the sequential decision problem, the agent uses its own policy  $\pi$  to make a sequence of decisions  $q_t, t = 1, \dots, T$ , based on the state it observed  $x_t, i = 1, \dots, T$ . Specifically, at each time step  $t$ , the agent is aware of the state information  $x_t$ . Then it uses its policy to take an action  $\pi(x_t) = q_t$ . The action  $q_t$  might have some influences on the environment of the agent, i.e., the next state  $x_{t+1}$  is influenced by  $q_t$ .

Modeling the tree search process as a sequential decision problem is a common method in machine learning [30], [31]. The procedure of the branch-and-bound algorithm, essentially a tree search process, is a sequential decision problem because each policy should make a decision at each iteration sequentially. Specifically, when learning the pruning policy, the state  $x_t$  is the feature of node  $N_t$ , e.g.,  $z_t^*$  and  $a_t^*$ , the action is whether to prune this node. Thus, learning the pruning policy is to learn a binary classifier.

Learning policies in sequential decision problems is often handled by reinforcement learning or imitation learning. The first paradigm, *reinforcement learning* [32], has been successfully applied in many promising applications such as robot manipulation [32], Playing Go [30], and game playing [31]. However, it has difficulty in dealing with sparse reward signals [26], which is common in learning MINLP problems. Furthermore, deep reinforcement learning-based tree search often takes long time to converge [30]. This will result in long training time especially for those whose relaxed problems take long time to solve, e.g., mixed-integer semidefinite programming (MISDP) and mixed-integer second order cone programming (MISOCP). We instead turn to another paradigm, namely *imitation learning* [33]. The difference between reinforcement learning and imitation learning is that there is an optimal policy in imitation learning. The imitation learning algorithm mimics the optimal policy instead of maximizing the reward, which allows it to overcome the problem of the sparse reward signals.

Imitation learning consists of a sequential decision problem and an oracle, i.e., the optimal policy  $\pi^*$ . In the branch and bound algorithm, the optimal policy should obtain the optimal solution with minimal number of nodes expanded. The optimal pruning policy only preserves the node  $N_t$  where the feasible region  $\mathcal{G}_t$  contains the optimal solution to problem  $\mathcal{P}$ . We call these nodes *optimal nodes*. With the oracle, imitation learning is reduced to a supervised learning problem. For each node, we first use a feature mapping  $\phi$  to extract the features of the node and label them by the oracle. We form the training dataset using the features and labels  $(\phi(N_1), \pi^*(N_1)), \dots, (\phi(N_T), \pi^*(N_T))$ . Then a supervised learning algorithm, e.g., support vector machine or logistic regression, is trained based on the training dataset.

The learning algorithm will achieve the best performance if we put all the nodes of the branch-and-bound tree into the training dataset. However, this is impractical even for medium-size wireless networks, because there are an exponential number of nodes. Meanwhile, putting only part of the nodes will degrade

the performance. The solution is to first train a policy on part of the nodes and then improve the learning performance by correcting the mistakes it makes. This is the idea of data aggregation (DAgger) [34], which is an iterative algorithm. Specifically, at the first iteration, the learned policy  $\pi^{(0)}$  is a policy which randomly prunes the nodes. At each iteration, we have a learned policy  $\pi^{(i)}$  and replace the pruning policy in the standard branch-and-bound algorithm with the learned policy. We collect the features of all the nodes explored by this policy and their corresponding labels produced by the oracle. Then a new policy  $\pi^{(i+1)}$  is learned based on the dataset collected at this iteration, and it corrects the mistakes made by  $\pi^{(i)}$ . Such iterations repeat for  $M$  times and we choose the policy with the best performance on the validation dataset. The pseudo-code of DAgger and data collection algorithm for branch-and-bound are shown in Algorithm 2 and Algorithm 3, respectively. In the algorithm, we do not prune the optimal nodes because the optimal nodes are desired behavior and we do not want to miss any of them. During the test stage, we just perform the branch-and-bound algorithm with a learned pruning policy  $\pi^{(k)}$  to prune nodes.

---

**Algorithm 2** DAgger( $\pi^*$ )

---

```

1:  $\pi^{(0)} \leftarrow \pi_r, \mathcal{D} \leftarrow \{\}$ 
2: for  $k \leftarrow 1, \dots, M$  do
3:   for  $j \leftarrow 1, \dots, |\mathcal{P}|$  do
4:      $\mathcal{D}^{(kj)} \leftarrow \text{COLLECT}(\pi^*, \pi^{(k)}, \mathcal{P}_j)$ 
5:      $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}^{(kj)}$ 
6:   end for
7:    $\pi^{(k+1)} \leftarrow \text{train a classifier using data } \mathcal{D}$ 
8: end for
9: return best  $\pi^{(k)}$  on validation set

```

---

*Generalization to different settings:* As input and output dimensions of most learning algorithms, e.g., MLP, in the test stage must be the same as in the training stage, most of the end-to-end learning methods [8], [10], [11] cannot generalize to settings with larger problem sizes than those in the training dataset. On the other hand, the input and output dimensions of the policies in the branch-and-bound algorithm are invariant when the problem size changes. Thus, generalization to different problem sizes is not a problem when learning the policies in the branch-and-bound algorithm. Such generalization capability also relies on the design of features (shown in Section III-D), and will be tested via simulations.

*Few-shot learning via learning policies:* The pruning policy at each node is a binary classifier. To learn the pruning policy is thus much easier than directly learning the input-output mapping of an

**Algorithm 3** COLLECT ( $\pi^*$ ,  $\pi$ ,  $p$ )

---

```

1:  $\mathcal{L} \leftarrow \{N_1\}, \mathcal{D} \leftarrow \{\}, t \leftarrow 0$ 
2: while  $\mathcal{L} \neq \emptyset$  do
3:    $t \leftarrow t + 1$ 
4:    $N_t \leftarrow$  select a node from  $\mathcal{L}$ 
5:    $(\mathbf{a}_t^*, \mathbf{w}_t^*), z_t^* \leftarrow$  solve the relaxed problem of  $\mathcal{P}_t$ 
6:    $f \leftarrow \phi(N_t, \mathbf{a}_t^*, \mathbf{w}_t^*, z_t^*)$ 
7:   if  $\pi^*(f) = \text{preserve}$  or  $\pi(f) = \text{preserve}$  then
8:      $N_t^1, N_t^2 \leftarrow$  two children of  $N_t$ 
9:      $\mathcal{L} \leftarrow \mathcal{L} \cup \{N_t^1, N_t^2\}$ 
10:  end if
11:   $\mathcal{D} \leftarrow \mathcal{D} \cup \{f, \pi^*(f)\}$ 
12: end while
13: return  $\mathcal{D}$ 

```

---

optimization algorithm with high dimensional inputs and outputs. This allows us to use a much lighter-weighted learning algorithm, with which we can achieve few-shot learning.

### B. Implementation Considerations

While the general framework has been described in the last subsection, there are a few implementation considerations. For a wide range of MINLP problems, e.g., MISOCP or MISDP, solving relaxed problems is very time consuming. Thus, in the implementation, we propose two accelerated techniques for these MINLP problems.

1) *Look-up table to accelerate training:* We propose to build a lookup table to accelerate the training process. During the training stage, we have to go through the dataset and solve the relaxed problem on the binary search tree at each iteration. We build a lookup table to save the solutions of all the relaxed problems we solved. If we encounter a relaxed problem that has not been solved before, we solve it and save this problem instance and its solution into the lookup table. Otherwise, we directly extract its solution from the lookup table. This method is found to be very effective throughout the simulations.

2) *Accelerated algorithm for binary problems:* We propose an accelerated algorithm for binary problems. Because the time for solving one instance of SOCP or SDP is hundreds or thousands of that of running one instance of neural networks, the computational complexity is dominated by solving relaxed problems in the framework. Thus, we accelerate the algorithm if we can reduce the numbers of relaxed

problems to solve. The solutions to the relaxed problems at non-leaf nodes are served as two purposes. The first is to be the features of the pruning policy, which is not important when other features are good enough. The second is to divide the feasible region, which is not needed in binary problems. Thus, to avoid solving the relaxed problems at the non-leaf nodes, the solution and objective value will not be included as the features for learning. This algorithm reduces the time complexity compared with the basic imitation learning framework by avoiding solving relaxed problems at non-leaf nodes.

### C. Neural Networks as Classifiers

In this paper, we propose to use neural networks as classifiers. A unique advantage of neural networks is that we can tune a parameter to control the computation-performance trade-off. This will not only enable us to use an iterative algorithm to guarantee the feasibility and outperform non-optimal labels, but also assist the development of the transfer learning approach (shown in Section IV-B).

Recall that the pruning policy is a binary classification problem, where the input is the feature vector of the node and the output is a binary class in  $\{prune, preserve\}$ . We employ MLP to do binary classification. Assuming that an  $L$ -layer MLP, a type of neural networks, is used, the  $k$ -th layer's output is calculated as:

$$\mathbf{g}^k = \text{Relu}(\mathbf{W}^k \mathbf{g}^{k-1} + \mathbf{b}^k), \quad (5)$$

where  $\mathbf{W}^k$  and  $\mathbf{b}^k$  are the learned parameters of the  $k$ -th layer.  $\mathbf{g}^k, k = 1, \dots, L$ , denotes the output of the  $k$ -th layer and  $\mathbf{g}^0$  is the input feature vector.  $\text{Relu}(\cdot)$  is the rectified linear unit function, i.e.,  $\max(0, \cdot)$ . The output indicates the probability of each class, which is a normalization of the last layer's output  $\mathbf{g}^L \in \mathbb{R}^2$ :

$$e[i] = \frac{\exp(\mathbf{g}^L[i])}{\sum_{j=1,2} \exp(\mathbf{g}^L[j])}, i = 1, 2, \quad (6)$$

where  $e[i]$  indicates the  $i$ -th component of vector  $e$ .

In the training stage, the loss function is the weighted cross entropy given by:

$$\ell = - \sum_{j=1,2} \mathbf{w}[j] \cdot \mathbf{y}[j] \log(e[j]), \quad (7)$$

where  $\mathbf{y}$  is the label, i.e.,  $\mathbf{y} = (1, 0)$  indicates that it belongs to the class *pruning*, and  $\mathbf{y} = (0, 1)$  otherwise.  $\mathbf{w}$  denotes the weight of each class, which is tuned by hand. Two parts contribute to the weight. Firstly, if the number of non-optimal nodes is much larger than the number of optimal nodes,

we should assign a higher weight to the class *preserve* in order to let the neural network not ignore this class. We denote this part as  $\mathbf{o}_1$  and it can be computed by:

$$\begin{aligned}\mathbf{o}_1[1] &= \frac{\# \text{ optimal nodes in the training set}}{\# \text{ nodes in the training set}}, \\ \mathbf{o}_1[2] &= 1 - \mathbf{o}_1[1].\end{aligned}\tag{8}$$

Secondly, when the number of feasible solutions is small, we should assign a higher weight to optimal nodes in the training dataset in order not to miss good solutions. This parameter is tuned by hand to achieve good performance on the validation dataset. We denote this part as  $\mathbf{o}_2$ . The total weight is calculated by  $\mathbf{w} = \mathbf{o}_1 \odot \mathbf{o}_2$ , where  $\odot$  is a hadamard product.

1) *Control computational complexity and guarantee feasibility*: Note that the MLP outputs  $\mathbf{e}$ , which indicates the probability of each class, followed by setting a threshold indicating which class it belongs to. We can set a threshold for pruning to control the computation-performance trade-off. Specifically, a pruning policy with a higher threshold will preserve more nodes than that with a lower threshold. Thus, at the test stage, we can set a lower threshold for a higher computational efficiency or a higher threshold for better performance. To guarantee the feasibility is a challenging problem in machine learning [7]. The key idea used here to address this problem is to iteratively increase the search space if not feasible. We view the infeasibility as the worst performance in an optimization problem. Therefore, the method used for controlling performance can be applied to guarantee the feasibility. Specifically, we iteratively increase the threshold for pruning if the problem is not feasible. We can eventually guarantee the feasibility in this way. To avoid long running time, we stop the algorithm after several iterations, which might cause some performance losses. However, in simulations, we find that the iteration to guarantee the feasibility is usually less than 2 when the training data distribution is the same as the test data.

2) *Outperform the non-optimal labels*: The threshold can also be utilized for outperforming the labels when the labels are not optimal solutions. By setting a higher threshold for pruning, more “good” nodes are explored and more “good” solutions are found. The output is the best solution among these solutions. Thus, it is possible that there is a solution having better performance than the labels. This methodology has proven to be effective throughout simulations (shown in Section VI) and will be used as a key component in LORA-TL (shown in Section IV-B).

#### D. Feature Design

The feature vector  $\mathbf{g}^0$  plays a critical role in the classifier, which should be carefully designed by leveraging domain knowledge. The available information consists of the problem data and the structure of the binary search tree. The features we used include two parts: the problem-independent features

and the problem-dependent features, which correspond to the structure of the tree and problem data respectively.

1) *Problem-independent Features*: This kind of feature only contain information about the branch-and-bound tree and the solutions and objective values of the relaxed problems. They are universal to all MINLP problems and are invariant from problems to problems. Thus, the learning framework will has good generalization capability if we only use problem-independent features. Such features can be categorized into three groups:

- 1) Node features, computed merely from the current node  $N_i$ , which contain the depth, the plunge depth of  $N_i$  and the optimal objective value of the relaxed problem  $z_i$ .
- 2) Branching features, computed from the branching variable, which contain the branching variable's value at the current node, i.e.,  $a_i^*[j]$ , that in the relaxed problem at its father node, and that in the relaxed problem of the root node, i.e.,  $a_1^*[j]$ .
- 3) Tree features, computed from the branch-and-bound search tree, which contain the optimal objective value at the root node, i.e.,  $z_1$ , the number of solutions found ever, and the best objective value found ever, i.e.,  $c^*$ .

Due to the significant variations among the objective value of  $\mathcal{P}$  under different network settings, all the objective values used as features in the branch-and-bound search tree should be normalized by the optimal objective value of the relaxed problem at the root node.

2) *Problem-dependent Features*: Problem-independent features are universal to all MINLP problems. Unfortunately, we cannot learn efficient policies in the branch-and-bound algorithm with only problem-independent features [35]. Thus, problem-dependent features, which will change for different problems, play an important role in learning effective policies. In a general setting of MINLP problems, the problem-dependent features can be the coefficients of integer variables. Narrowing down to the problems in wireless communications, the problem dependent features can be channel state information (CSI) or a function of CSI. The problem dependent features for interference channel power management and the network power minimization in Cloud-RAN are described in Section VI. Since the problem-dependent features depend on the size and distribution of problem data, they should be carefully designed for a good generalization capability. Commonly used approaches to handle the generalization for problem-dependent features are the normalization of the features or attention mechanism [36].

#### E. Summary of LORA

Overall, LORA has the following advantages over end-to-end learning:

- 1) It achieves near-optimal performance with few training samples;

- 2) It guarantees feasibility of constraints without performance loss;
- 3) It outperforms the non-optimal labels;
- 4) It generalizes to different system configurations, and is able to scale up to larger problem sizes.

#### IV. LORA-TL: TRANSFER LEARNING VIA SELF-IMITATION

While the imitation learning framework helps to achieve few-shot learning for MINLP resource allocation problems, its practical implementation still faces a few key challenges, which will be illustrated in this section. Then we propose a transfer learning method to address these issues.

##### A. Limitations of the Imitation Learning Framework

An essential assumption of most machine learning algorithms is that the training and future testing data are in the same feature space with the same distribution, i.e., the same task [27]. Narrowing down to problems in wireless networks, the “distribution” includes “structure”, e.g., large-scale fading and signal-to-noise-ratio (SNR), and “size”, e.g., the numbers of users, base stations, and antennas. The performance deteriorates when the machine learning task to be applied is different from the trained one, which is referred to as the *task mismatch* issue. How much will the performance deteriorate is determined by the similarity between the tasks, which is little known and on-going research problems in machine learning [37]. A straightforward way to resolve this issue is to collect enough additional training samples and labels and to train a neural network for the new setting from scratch. This will achieve good performance, but it is impractical in real systems because training neural networks requires large amounts of samples and long computing time. To cope with the dynamics of wireless networks, it is highly desirable to reduce the training time for the new task, i.e., when the network setting changes. Note that although the tasks are different, they share something in common, i.e., the same structure of the underlying optimization problem. In other words, the knowledge learned in the old task can be helpful for the new task. Thus, it is possible to train a new model with only a few additional samples if we can effectively transfer such knowledge. This can significantly reduce the training time and achieve good performance in the new task. The difference between traditional machine learning and transfer learning is shown in Fig. 2.

##### B. Self-imitation with Unlabeled Samples

In this subsection, we first present transfer learning via fine-tuning and discuss its limitations. Then we propose an approach called *self-imitation* to address the problems for fine-tuning.

Fine-tuning is the most frequently employed method for transfer learning in neural networks. Neural networks are usually trained via stochastic gradient descent (SGD). For different layers, we can have



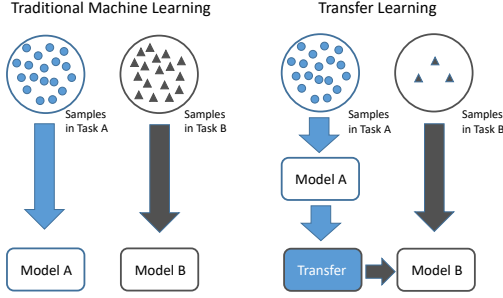


Fig. 2. The key difference between transfer learning and traditional machine learning is that transfer learning can tackle the task mismatch problem with few additional training samples. This is achieved by transferring the knowledge from the old task into the new task.

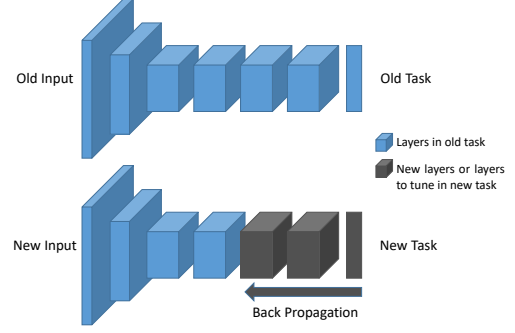


Fig. 3. An illustration of transfer learning via fine-tuning. The neural network for the new task keeps the first several layers from the neural network for the old task. The last several layers are trained from scratch or tuned from those for the old task with a small learning rate.

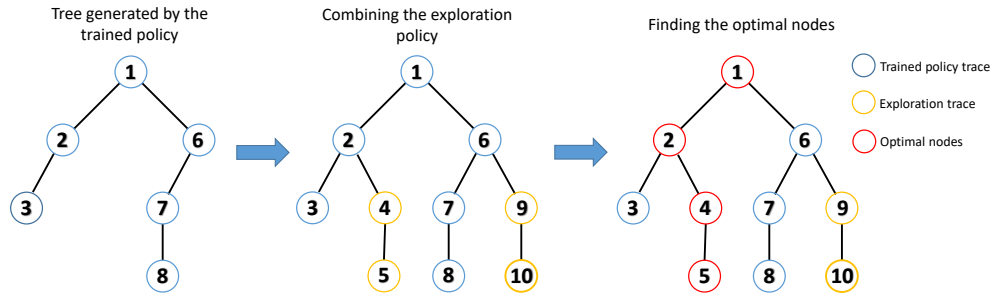


Fig. 4. An illustration of the self-exploration algorithm. The number represents the order of exploration. The nodes explored in the branch-and-bound tree for the additional training samples in the new scenario include two parts: the nodes explored by the policy trained on the original dataset and the nodes explored by the exploration policy. Then the best solution found during the exploration will be used to mark optimal nodes for training neural networks.

different learning rates, i.e., the step size of the SGD. Fine-tuning is to tune the learning rate of each layer to refine the pre-trained neural network on the additional training dataset. We can train the pre-trained neural network with a small learning rate. The knowledge learned on the original dataset can serve as a good initialization point. The learning rate is small because we expect that the initial weights are relatively good, so distorting them too much and too quickly is not a smart choice. The illustration of fine-tuning is shown in Fig. 3. Fine-tuning reduces the training time, but it needs additional labeled samples. The time cost is still expensive as the computational complexity of branch-and-bound to generate the labels, i.e., the optimal solutions, is exponential. This implies we even have difficulty in generating a small amount of training labels if the network size is large. Thus, it will be desirable if we can refine

the model with unlabeled data.

---

**Algorithm 4** Self-Imitation( $\pi$ )

---

```

1:  $\mathcal{D} \leftarrow \{\}$ 
2: for  $k \leftarrow 1, \dots, M$  do
3:   for  $j \leftarrow 1, \dots, |\mathcal{P}|$  do
4:      $\hat{\pi}^{(k)} \leftarrow \pi^{(k)} + \pi_e$ 
5:      $\mathcal{D}^{(kj)} \leftarrow \text{COLLECT-SI}(\pi^{(k)}, \mathcal{P}_j)$ 
6:      $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}^{(kj)}$ 
7:   end for
8:    $\pi^{(k+1)} \leftarrow \text{fine-tune the classifier } \pi^{(k)} \text{ using data } \mathcal{D}$ 
9: end for
10: return best  $\pi^{(k)}$  on validation set

```

---

We propose a transfer learning via self-imitation to address the task mismatch problem with a few additional unlabeled samples. The old policy  $\pi$  was trained on the original training dataset. As the task changes, we can collect a few additional unlabeled samples. The learned policy may perform badly on these additional training samples. We blend  $\pi$  with an exploration policy  $\pi_e$ . The blended policy will have better performance than the learned policy because the search space is larger. Then we use the solution produced by the blended policy as the labels and fine-tuning the classifier on the additional training samples and labels. The performance of the labels will be improved iteratively by repeating such a process, and thus the performance of the learned policy will be improved iteratively. The pseudo-code of the iterative training algorithm is shown as Algorithm 4.

The data collection algorithm is different from those in DAgger (shown in Algorithm 2), since there is no optimal policy to produce training labels. We instead get labels by examining its past decisions. Specifically, we first perform a standard branch-and-bound procedure except using the blended policy  $\hat{\pi}^{(k)}$  to prune nodes. The node features are collected without labels. We also record the best solution found  $c^*$  and its corresponding node  $N_{opt}$  during the process. Then  $c^*$  is used as the oracle for imitation learning. We label all the nodes on the path from the root node to  $N_{opt}$  as the optimal nodes and add these labels into the aggregation dataset  $\mathcal{D}^{(kp)}$ . The pseudo-code for data collection in self-imitation is shown in Algorithm 5 and the exploration is illustrated in Fig. 4.

*Exploration Policy:* The key for Algorithm 4 is an efficient exploration policy  $\pi_e$ . Recall that in Section III-C, we propose to employ the MLP to learn the pruning policy and set a higher threshold for the class *prune* to outperform non-optimal labels. Our proposed exploration policy is the learned policy with a

---

**Algorithm 5** COLLECT-SI ( $\pi, p$ )
 

---

```

1:  $\mathcal{L} \leftarrow \{N_1\}, \mathcal{D} \leftarrow \{\}, t \leftarrow 0, \mathcal{N}_{opt} \leftarrow \{N_1\}$ 
2: while  $\mathcal{L} \neq \emptyset$  do
3:    $t \leftarrow t + 1$ 
4:    $N_t \leftarrow$  select a node from  $\mathcal{L}$ 
5:    $(\mathbf{a}_t^*, \mathbf{w}_t^*), z_t^* \leftarrow$  solve the relaxed problem of  $\mathcal{P}_t$ 
6:    $f \leftarrow \phi(N_t, \mathbf{a}_t^*, \mathbf{w}_t^*, z_t^*)$ 
7:   if  $\pi(f) = \text{preserve}$  then
8:      $N_t^1, N_t^2 \leftarrow$  two children of  $N_t$ 
9:      $\mathcal{L} \leftarrow \mathcal{L} \cup \{N_t^1, N_t^2\}$ 
10:  end if
11:  if  $a_t^*[j], \forall j$  is integer then
12:     $c^* \leftarrow \min(c^*, z_t^*)$ 
13:     $N_{opt} \leftarrow N_t$ 
14:  end if
15:   $\mathcal{D} \leftarrow \mathcal{D} \cup \{f\}$ 
16: end while
17: while  $N_{opt} \neq N_1$  do
18:    $\mathcal{N}_{opt} \leftarrow \mathcal{N}_{opt} \cup \{N_{opt}\}$ 
19:    $N_{opt} \leftarrow \text{father}(N_{opt})$ 
20: end while
21: for  $j \leftarrow 1, \dots, |\mathcal{D}|$  do
22:   if  $\text{node}(\mathcal{D}_j) \in \mathcal{N}_{opt}$  then
23:      $\mathcal{D}_j \leftarrow \{\mathcal{D}_j.f, \text{preserve}\}$ 
24:   else
25:      $\mathcal{D}_j \leftarrow \{\mathcal{D}_j.f, \text{prune}\}$ 
26:   end if
27: end for
28: return  $\mathcal{D}$ 

```

---

higher threshold for *prune*. Thus, as is shown in Section III-C, this exploration policy is to achieve better performance on the new dataset to serve as the labels for fine-tuning. It is more efficient than traditional exploration policies, e.g.,  $\epsilon$ -greedy policy [32], because the trained policy has a prior knowledge on the underlying optimization problem structure.

## V. APPLICATION EXAMPLES

To evaluate their effectiveness, we test LORA and LORA-TL on two resource allocation problems in wireless networks, i.e., interference channel power control and network power minimization. This section presents the formulations of the two problems, and the next section presents the numerical results.

### A. Interference Channel Power Control

The first problem is power control for interference management in a  $K$ -user single-antenna interference channel [28]. The received signal at the  $k$ -th receiver is given by

$$y_k = h_{kk}s_k + \sum_{j \neq k} h_{kj}s_j + n_k,$$

where  $h_{kk} \in \mathbb{C}$  denotes the direct-link channel between the  $k$ -th transmitter and  $k$ -th receiver,  $h_{kj} \in \mathbb{C}$  denotes the cross-link channel between transmitter  $j$  and receiver  $k$ ,  $s_k \in \mathbb{C}$  denotes the data symbol for the  $k$ -th receiver, and  $n_k \sim \mathcal{CN}(0, \sigma_k^2)$  is the Gaussian additive noise independent with  $s_k$ .

The signal-to-interference-plus-noise ratio (SINR) for the  $k$ -th receiver is given by

$$\text{SINR}_k = \frac{|h_{kk}|^2 p_k}{\sum_{i \neq k} |h_{ki}| p_i + \sigma_k^2},$$

where  $p_k = \mathbb{E}[s_k^2]$  is the power of the  $k$ -th transmitter, and  $0 \leq p_k \leq P_{\max}$ .

Denote  $\mathbf{p} = [p_1, \dots, p_K]$  as the power allocation vector. The objective is to find the optimal power allocation for all users to maximize the sum rate, and the formulation is given by

$$\begin{aligned} & \underset{\mathbf{p}}{\text{maximize}} && \sum_{k=1}^K \log_2 \left( 1 + \frac{|h_{kk}|^2 p_k}{\sum_{i \neq k} |h_{ki}| p_i + \sigma_k^2} \right) \\ & \text{subject to} && 0 \leq p_k \leq P_{\max}, \forall k. \end{aligned} \tag{9}$$

This problem is non-convex and NP-hard [28]. It has been discovered that binary power control, i.e., to take  $p_k$  either as 0 or  $P_{\max}$ , achieves considerably good performance [38], which will be adopted in our evaluation. After replacing the box constraints with binary constraints  $p_k \in \{0, P_{\max}\}$ , this problem is turned into an interger programming problem and can be solved by the branch-and-bound algorithm.

### B. Network Power Minimization in Cloud-RANs

The second example is network power minimization problem in Cloud-RANs [5]. This is a typical MINLP problem in wireless networks as it consists of discrete variables (i.e., the selection of RRHs and fronthaul links), continuous variables (i.e., downlink beamforming coefficients), and QoS constraints and power constraints.

Consider a Cloud-RAN with  $L$  remote radio heads (RRHs) and  $K$  single-antenna mobile users (MUs), where the  $l$ -th RRH is equipped with  $N_l$  antennas. The baseband unit (BBU) pool is connected to all the RRHs via a high-bandwidth, low-latency fronthaul network, and performs the centralized signal processing. The corresponding SINR for the  $k$ -th MU is given by

$$\text{SINR}_k = \frac{|\sum_{l \in \mathcal{L}} \mathbf{h}_{kl}^H \mathbf{w}_{lk}|^2}{\sum_{i \neq k} |\sum_{l \in \mathcal{L}} \mathbf{h}_{kl}^H \mathbf{w}_{li}|^2 + \sigma_k^2}, \forall k \in \mathcal{S},$$

where  $\mathbf{w}_{lk} \in \mathbb{C}^{N_l}$  denotes the transmit beamforming vector from RRH  $l$  to MU  $k$ ,  $\mathbf{h}_{kl} \in \mathbb{C}^{N_l}$  denotes the channel vector between the  $k$ -th MU and the  $l$ -th RRH, and  $\sigma_k^2$  is the variance of additive noise [5].

Let a binary vector  $\mathbf{a} = (a_1, \dots, a_L)$  with  $a_i \in \{0, 1\}$  denote the mode of each RRH, i.e.,  $a_i = 1$  (resp.  $a_i = 0$ ) if the  $i$ -th RRH and the corresponding transport link are switched on (resp. switched off). Each RRH has its own transmit power constraint

$$\sum_{k=1}^K \|\mathbf{w}_{lk}\|_2^2 \leq a_l \cdot P_l, l \in \{1, \dots, L\},$$

where  $\|\cdot\|_2$  is the  $\ell_2$ -norm of a vector.

The network power consumption in Cloud-RAN consists of the relative fronthaul network power consumption and the total transmit power consumption [5]. Thus, the network power minimization problem is formulated as the following MINLP problem:

$$\begin{aligned} & \underset{\mathbf{w}, \mathbf{a}}{\text{minimize}} && \sum_{l=1}^L a_l \cdot P_l^c + \sum_{l=1}^L \sum_{k=1}^K \frac{1}{\eta_l} \|\mathbf{w}_k\|_2^2 \\ & \text{subject to} && \sqrt{\sum_{i \neq k} |\mathbf{h}_k^H \mathbf{w}_i|^2 + \sigma_k^2} \leq \frac{1}{\gamma_k} \Re(\mathbf{h}_k^H \mathbf{w}_k), \forall k \\ & && \sqrt{\sum_{k=1}^K \|\mathbf{A}_{lk} \mathbf{w}_k\|_2^2} \leq a_l \cdot \sqrt{P_l}, \forall l \\ & && a_l \in \{0, 1\}, \forall l, \end{aligned} \tag{10}$$

where  $P_l^c$  is the relative fronthaul link power consumption [5], i.e., the power saved when both the RRH and the corresponding fronthaul link are switched off,  $\eta_l$  is the drain efficiency of the radio frequency power amplifier, the aggregative beamforming vector is  $\mathbf{w} = [\mathbf{w}_1^T, \dots, \mathbf{w}_K^T]^T \in \mathbb{C}^{NK}$  with

$\mathbf{w}_k = [\mathbf{w}_{1k}^T, \dots, \mathbf{w}_{Lk}^T]^T \in \mathbb{C}^N$  and  $N = \sum_{l=1}^L N_l$ ,  $\mathbf{h}_k = [\mathbf{h}_{1k}^T, \dots, \mathbf{h}_{Lk}^T]^T \in \mathbb{C}^N$ ,  $\Re(\cdot)$  denotes the real part of a complex scalar, and  $\mathbf{A}_{lk} \in \mathbb{C}^{N \times N}$  is a block diagonal matrix with an identity matrix  $\mathbf{I}_{N_l}$  as the  $l$ -th main diagonal block matrix and zeros elsewhere.

This problem is reduced to SOCP if the integer variable  $\mathbf{a}$  is fixed. Thus, it belongs to the problem class  $\mathcal{P}$  and can be solved by LORA.

## VI. SIMULATIONS

In this section, we evaluate the performance of LORA and LORA-TL via two examples introduced in the last section. The simulations include three aspects:

- 1) The performance of the LORA.
- 2) The generalization ability of LORA.
- 3) The ability of the LORA-TL to tackle the task mismatch issue.

### A. Interference Channel Power Control

We consider the symmetric Rayleigh fading interference channel model, i.e., the channel of each link is i.i.d. as  $h_{ij} \sim \mathcal{CN}(0, 1)$ . Without loss of generality, we assume  $P_{\max} = 1$  and define SNR as  $\text{SNR} = 10 \log_{10} \left( \frac{P_{\max}}{\sigma^2} \right)$ .

In this problem, we adopt the depth first policy for node selection and variable selection. For problem-independent features, we use the branching variable's value  $a_i[j]$ . Intuitively, we encourage the links with large channel gains and small interference to be active. Thus, the problem-dependent features for the node  $N_i$  with branch variable index  $j$  include: the channel gain  $|h_{jj}|^2$  and the reverse ranking of it in  $\{|h_{kk}|^2, k = 1, \dots, K\}$ ; the two aggregated interference terms  $\sum_{k \neq j} \frac{|h_{ji}|^2}{K-1}$  and  $\sum_{k \neq j} \frac{|h_{ij}|^2}{K-1}$ . We use a 3-layer MLP and set the number of hidden units as  $\{16, 32, 16\}$ , i.e., the first layer has 16 hidden units, the second layer has 32 hidden units, and the third layer has 16 hidden units. We set the maximal iteration of DAgger as  $M = 1$  and train the neural network for 25 epochs in this example.

There are three benchmarks:

- 1) Greedy Binary Power Control (GBPC) [38]: This method is an approximation of the branch-and-bound algorithm in the setting of binary power control. It is shown in [38] that this method approaches the performance of optimal binary power control.
- 2) WMMSE [9]: This method is to solve the sum-utility maximization problem in the MIMO interfering channel. It uses block coordinate descent and converges to the stationary point of the simulated problem. It was proposed in [8] to employ MLP to approximate the solution of this algorithm.
- 3) Branch-and-Bound algorithm (Optimal).

1) *Performance of LORA*: We first demonstrate the near-optimal performance of LORA. The labels, i.e., the optimal solutions, are generated by the standard branch-and-bound algorithm. In this part, we consider  $K = 10$  under different SNR values. For each setting in Table II, we generate 200 samples for training and 200 samples for test. The sum rates of the proposed algorithm and other competing methods (normalized by the sum rate of WMMSE) are shown in Table II. The comparison shows that LORA achieves near-optimal performance and outperforms other competing methods when trained on 200 samples. As a comparison, for the same problem size,  $10^8$  and 20000 training channel samples are used in [10] and [11] respectively, which shows that our proposed method is extremely sample-efficient.

TABLE II  
THE SUM RATE OF LORA AND OTHER COMPETING METHODS (NORMALIZED BY THE SUM RATE OF WMMSE).

Setting	Optimal	LORA	GBPC	WMMSE
$K = 10, \text{SNR} = 0$	102.3%	101.8%	101.5%	100%
$K = 10, \text{SNR} = 5$	104.0%	103.4%	101.7%	100%
$K = 10, \text{SNR} = 10$	104.4%	103.8%	100.4%	100%
$K = 10, \text{SNR} = 20$	107.8%	106.4%	104.1%	100%

We then investigate the performance of LORA when the samples are not labeled by the optimal solutions. Instead, we use the WMMSE algorithm to generate labels for learning, as branch and bound cannot be applied for large-size problems. For each setting, we generate 500 samples for training and 200 samples for test. We set the threshold for pruning as 0.9 during the test. The performances of LORA and other competing methods are shown in Table III. We see that the proposed imitation learning framework not only outperforms its “teacher”, i.e., WMMSE, but also outperforms GBPC when  $\text{SNR} \geq 5\text{dB}$ . This result indicates that we may use heuristic algorithms to generate training labels, which can accelerate the training process. It also shows that unlike other supervised learning methods, the performance of LORA is not be upper bounded by the label.

2) *Generalization of LORA*: The results above show the superiority of LORA when the training and test settings are the same, but they may be different in practice, e.g., the numbers of the users are constantly changing. This requires the generalization ability of LORA, which is tested next. We train on 200 samples with  $K = 10$  and  $\text{SNR} = 5$ , and test it on different numbers of users and different SNRs. The results are shown in Table IV. From the table, we see that LORA trained on  $K = 10$  and  $\text{SNR} = 5$  outperforms other competing methods and achieves comparable results to “LORA (full training)”, which

TABLE III  
THE SUM RATE OF LORA WHEN THE TRAINING LABELS ARE NOT OPTIMAL (NORMALIZED BY THE SUM RATE OF WMMSE).

Setting	LORA	GBPC	WMMSE
$K = 20, \text{SNR} = 0$	100.6%	100.8%	100%
$K = 20, \text{SNR} = 5$	101.5%	100.5%	100%
$K = 20, \text{SNR} = 10$	103.0%	100.4%	100%
$K = 20, \text{SNR} = 20$	106.4%	103.9%	100%

trains the model on the test setting. This demonstrates that LORA can handle moderately variation of the system configurations.

TABLE IV  
THE SUM RATE OF LORA WHEN THE TRAINED SETTING IS DIFFERENT (NORMALIZED BY THE SUM RATE OF WMMSE).

Setting	LORA <sup>3</sup> (full training)	LORA (train on $K = 10, \text{SNR}=5$ )	GBPC
$K = 10, \text{SNR} = 10$	103.8%	103.8%	100.8%
$K = 10, \text{SNR} = 20$	106.4%	106.3%	104.1%
$K = 20, \text{SNR} = 10$	103.0%	102.7%	100.4%
$K = 20, \text{SNR} = 20$	106.4%	105.7%	103.9%

3) *The performance of LORA-TL:* At last, we verify the effectiveness of LORA-TL. We first train LORA on 100 labeled samples with  $K = 5, \text{SNR} = 0$ . Its direct generalization to different settings is shown in Table V as “LORA (train on  $K = 5, \text{SNR} = 0$ )”. Significant performance loss is shown, as the test setting is dramatically different from the trained one, and this result illustrates the need for the LORA-TL framework. For LORA-TL, we transfer the model on 50 unlabeled samples with the same system setting as the test setting. We set the number of iterations of self-exploration as  $M = 10$  and the threshold for pruning as 0.95 in each iteration. The results are shown in Table V as “LORA-TL”. “LORA (full training)” is to train with LORA on sufficient labeled samples whose system configuration is the same as the test dataset. We see LORA-TL outperforms competing methods and achieves comparable results with “LORA (full training)”. This demonstrates LORA-TL can adapt to different system configurations with only tens of additional unlabeled training samples, which is attractive for practical uses.

<sup>3</sup>For LORA “full training”, we use the the solutions by the branch-and-bound when  $K = 10$  and by WMMSE as labels when  $K = 20$ .



TABLE V  
THE SUM RATE OF THE PROPOSED ALGORITHM AND OTHER COMPETING METHODS

Setting	LORA (train on $K = 5$ , SNR = 0)	LORA-TL	LORA (full training)	GBPC	WMMSE
$K = 10$ , SNR = 10	89.2%	102.9%	103.8%	100.8%	100%
$K = 10$ , SNR = 20	89.3%	105.8%	106.4%	104.1%	100%
$K = 20$ , SNR = 10	81.9%	101.7%	103.0%	100.4%	100%
$K = 20$ , SNR = 20	66.1%	104.8%	106.4%	103.9%	100%

### B. Network Power Minimization in Cloud-RANs

For network power minimization in Cloud-RANs, we consider the channel model  $\mathbf{h}_{kl} = 10^{-L(d_{kl})/20} \sqrt{\phi_{kl} s_{kl}} \mathbf{g}_{kl}$ , where  $L(d_{kl})$  is the path-loss at distance  $d_{kl}$ ,  $\phi_{kl}$  is the antenna gain,  $s_{kl}$  is the shadowing coefficient, and  $\mathbf{g}_{kl}$  is the small scale fading coefficient. One can refer to Table I in [5] for the detailed parameter setting.

We set the hidden units as  $\{32, 64, 16\}$ . We use the branching variable's value, i.e.,  $a_i[j]$  and its value at the root problem, i.e.,  $a_0[j]$ , as the problem-independent features. As shown in [5], we encourage to open a small number of RRHs and RRHs with small fronthaul link power consumption if the problem is feasible. Thus, the problem-dependent features are selected as the fronthaul link power consumption, i.e.,  $\frac{P_j^c \cdot L}{\sum_{k=1}^L P_k^c}$ . We adopt the accelerated algorithm introduced in Section III-B. In addition, we use the iterative algorithm to guarantee the feasibility introduced in Section III-C, and set the threshold for preserving a node as  $0.5 \cdot 0.8^l$ , where  $l$  is the number of iterations of the iterative algorithm. We set the number of DAgger as  $M = 5$  and train 5 epochs for each DAgger iteration.

There are three benchmarks:

- 1) Relaxed Mixed-integer Nonlinear Programming based algorithm (RMINLP) [39]: In this algorithm, the deflation procedure is performed to switch off RRHs one-by-one based on the solution to a relaxed problem.
- 2) Iterative Group Sparse Beamforming (GSBF) [5]: This is the state-of-art algorithm in the network power minimization in Cloud-RAN.
- 3) Branch-and-Bound algorithm (Optimal solution).

1) *Performance of LORA*: We simulate the performance under the setting with the target SINR (TSINR) as 4dB, with 10 2-antenna RRHs and different numbers of users. The RRHs and MUs are uniformly and independently distributed in the square region  $[-1000, 1000] \times [-1000, 1000]$  meters. The fronthaul link power consumption is set to be a random permutation of  $P_l^c = (5 + l)W, l = 1, \dots, 10$  in

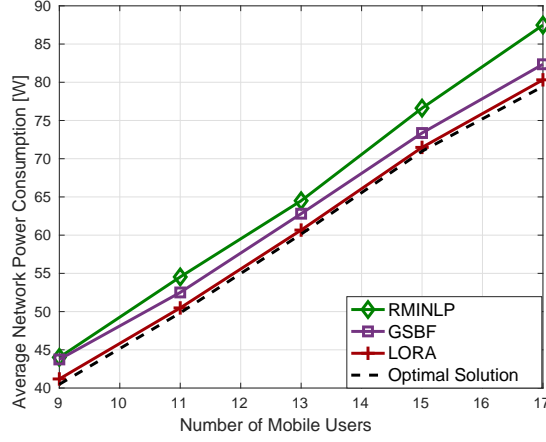


Fig. 5. The average network power consumption versus the number of mobile users.

different samples. For each setting, we generate 50 network realizations, i.e., samples, for training and 50 network realizations for test. The performance is shown in Fig. 5. It is shown that LORA outperforms the specialized state-of-art methods in the network power minimization problem and achieves near-optimal performance. The locations of RRHs and MUs in the training dataset and test dataset are different, and thus the results already demonstrate certain generalization capability of LORA, i.e., generalization to different large-scale fading.

TABLE VI  
THE GAP TO THE OPTIMAL OBJECTIVE VALUE.

Setting	LORA (full training)	LORA (train on $L = 6, K = 9, \text{TSINR} = 0$ )	RMINLP	GSBF
$L = 10, K = 7$	1.87%	4.39%	7.94%	12.9%
$L = 10, K = 9$	1.73%	2.97%	8.71%	8.04%
$L = 10, K = 11$	1.30%	1.72%	9.44%	5.36%
$L = 10, K = 13$	0.89%	0.89%	7.27%	4.45%
$L = 10, K = 15$	0.70%	1.48%	7.94%	3.36%

2) *Generalization of LORA*: We conduct the following experiment to test the generalization beyond different large-scale fading. We first train the model on 100 samples with  $L = 6, K = 9, \text{TSINR} = 0$ , and test it on  $L = 10, \text{TSINR} = 4$ , with different numbers of users. The results are shown in Table VI. From the table, we see that even if trained on a different setting, LORA achieves state-of-art performance, which demonstrates its good generalization ability.

3) *Performance of LORA-TL*: Throughout experiments, we observe a performance deterioration when the numbers of MUs change dramatically. In this part, we test the effectiveness of LORA-TL in this challenging situation, where LORA cannot achieve good performance. We conduct two experiments to test LORA-TL: when the numbers of MUs in the test dataset are much less and much larger than the training dataset. In the first experiment, we train LORA-TL on 100 labeled samples with  $L = 6$ ,  $K = 6$ ,  $\text{TSINR} = 0$ , transfer on 10 unlabeled samples with  $L = 10$ ,  $K = 15$ , and test on  $L = 10$ ,  $K = 15$ . The results are shown in Table VII. In the second experiment, we train LORA-TL on 50 labeled samples with  $L = 10$ ,  $K = 17$ ,  $\text{TSINR} = 4$ , transfer on 10 unlabeled samples with  $L = 10$ ,  $K = 7$ , and test on  $L = 10$ ,  $K = 7$ . For the self-imitation algorithm, the number of iterations for self-imitation is set as  $M = 10$ . The threshold for pruning is set as 0.9 at the first iteration and half if this iteration's performance is the same as last iteration. The results are shown in Table VIII. As shown in the tables, equipped with transfer learning, LORA-TL outperforms all competing methods and achieves comparable performance with LORA trained on sufficient samples of the same distribution with the test dataset.

TABLE VII  
THE GAP TO THE OPTIMAL OBJECTIVE VALUE.

Setting	LORA (train on $L = 6$ , $K = 6$ , $\text{TSINR} = 0$ )	LORA-TL	LORA (full training)	RMINLP	GSBF
$L = 10$ , $K = 15$ $\text{TSINR} = 0$	— <sup>4</sup>	0.63%	0.61%	7.07%	6.76%
$L = 10$ , $K = 15$ $\text{TSINR} = 2$	—	0.38%	0.3%	7.92%	5.06%
$L = 10$ , $K = 15$ $\text{TSINR} = 4$	—	0.68%	0.39%	7.94%	3.36%

### C. Summary of the Experiments

Overall, the extensive experiments in this section demonstrate that

- 1) LORA achieves near-optimal performance with only tens or hundreds of training samples;
- 2) LORA generalizes well when the difference between the system configurations of the training dataset and test dataset is moderate;

<sup>4</sup>To control the time during the test, we set the maximal number of iterations to guarantee the feasibility as 5. It only finds feasible solutions for 82% of the problems during the test. So we do not record its performance here.

TABLE VIII  
THE GAP TO THE OPTIMAL OBJECTIVE VALUE.

Setting	LORA (train on $L = 10$ , $K = 17$ , TSINR = 4)	LORA (full training)	LORA-TL	RMINLP	GSBF
$L = 10$ , $K = 7$ TSINR = 0	57.1%	0.72%	2.48%	7.43%	10.0%
$L = 10$ , $K = 7$ TSINR = 2	43.8%	2.30%	2.79%	5.57%	11.9%
$L = 10$ , $K = 7$ TSINR = 4	29.9%	1.87%	2.95%	6.89%	12.9%

- 3) When the difference is dramatic, with tens of additional unlabeled training samples, LORA-TL can achieve comparable performance with LORA trained on sufficient samples of the same distribution with the test dataset.

## VII. CONCLUSIONS

In this paper, we proposed two machine learning-based frameworks, namely LORA and LORA-TL, to achieve near-optimal performance for MINLP resource allocation problems in wireless networks. We first proposed LORA, an imitation learning-based approach, to learn the pruning policy in the optimal branch-and-bound algorithm. By exploiting the structure of the branch-and-bound algorithm, LORA achieves near-optimal performance with few training samples, guarantees constraint feasibility, is able to perform better than the labels, and generalizes to problem instances with larger sizes than the training dataset. Furthermore, we proposed LORA-TL to address the task mismatch issue for LORA, relying on only a few additional unlabeled training samples. Extensive experiments have been done to verify the effectiveness of both LORA and LORA-TL.

Different from previous approaches for “learning to optimize” that directly approximate the input-output mapping, our proposed frameworks achieve few-shot learning by exploiting the algorithm structure. As training samples are in general expensive to obtain, we expect these frameworks will facilitate the adoption of the machine learning-based methods in wireless communications. For the next stage, it will be interesting to test the proposed methods on other design problems in wireless networks, and develop approaches to further reduce the sample size.

## REFERENCES

- [1] Y. Shen, Y. Shi, J. Zhang, and K. B. Letaief, “Scalable network adaptation for Cloud-RANs: An imitation learning approach,” in *Proc. IEEE Global Conf. Signal Info. Process.*, Anaheim, CA, Dec. 2018.

- [2] Z. Han and K. R. Liu, *Resource allocation for wireless networks: basics, techniques, and applications*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
- [3] C. Y. Wong, R. S. Cheng, K. B. Lataief, and R. D. Murch, "Multiuser OFDM with adaptive subcarrier, bit, and power allocation," *IEEE J. Sel. Areas Commun.*, vol. 17, pp. 1747–1758, Oct. 1999.
- [4] Q. Ye, B. Rong, Y. Chen, M. Al-Shalash, C. Caramanis, and J. G. Andrews, "User association for load balancing in heterogeneous cellular networks," *IEEE Trans. Wireless Commun.*, vol. 12, pp. 2706–2716, Jun. 2013.
- [5] Y. Shi, J. Zhang, and K. B. Letaief, "Group sparse beamforming for green Cloud-RAN," *IEEE Trans. Wireless Commun.*, vol. 13, pp. 2809–2823, May 2014.
- [6] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, pp. 3590–3605, Dec. 2016.
- [7] B. Yoshua, L. Andrea, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d’horizon," *arXiv preprint arXiv:1811.06128*, 2018.
- [8] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for interference management," *IEEE Trans. Signal Process.*, vol. 66, pp. 5438 – 5453, Oct. 2018.
- [9] Q. Shi, M. Razaviyayn, Z. Luo, and C. He, "An iteratively weighted MMSE approach to distributed sum-utility maximization for a mimo interfering broadcast channel," *IEEE Trans. Signal Process.*, vol. 59, pp. 4331–4340, Sept. 2011.
- [10] F. Liang, C. Shen, W. Yu, and F. Wu, "Towards optimal power control via ensembling deep neural networks," *arXiv preprint arXiv:1807.10025*, 2018.
- [11] W. Lee, M. Kim, and D.-H. Cho, "Deep power control: Transmit power control scheme based on convolutional neural network," *IEEE Commun. Lett.*, vol. 22, pp. 1276–1279, Apr. 2018.
- [12] Y. S. Nasir and D. Guo, "Deep reinforcement learning for distributed dynamic power allocation in wireless networks," *arXiv preprint arXiv:1808.00490*, 2018.
- [13] W. Cui, K. Shen, and W. Yu, "Spatial deep learning for wireless scheduling," *arXiv preprint arXiv:1808.01486*, 2018.
- [14] M. Eisen, C. Zhang, L. F. Chamon, D. D. Lee, and A. Ribeiro, "Learning optimal resource allocations in wireless systems," *arXiv preprint arXiv:1807.08088*, 2018.
- [15] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, *et al.*, "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, p. 471, Oct. 2016.
- [16] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Adv. Neural Inform. Process. Syst.*, pp. 2692–2700, Dec. 2015.
- [17] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," in *Proc. Int. Conf. Learning Representation*, Apr. 2017.
- [18] H. He, H. Daume III, and J. M. Eisner, "Learning to search in branch and bound algorithms," in *Proc. Adv. Neural Inform. Process. Syst.*, pp. 3293–3301, Dec. 2014.
- [19] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Proc. Adv. Neural Inform. Process. Syst.*, pp. 6348–6358, Dec. 2017.
- [20] M. Kruber, M. E. Lübbecke, and A. Parmentier, "Learning when to use a decomposition," in *Proc. Int. Conf. AI OR Techniques Constraint Programming Combinatorial Optimization Problems*, pp. 202–210, Springer, Jun. 2017.
- [21] A. Cypher and D. C. Halbert, *Watch what I do: programming by demonstration*. MIT press, 1993.
- [22] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Proc. Adv. Neural Inform. Process. Syst.*, pp. 3981–3989, Dec. 2016.

- [23] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Trans. Neural Networks*, vol. 9, pp. 987–1000, Sept. 1998.
- [24] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.
- [25] S. Thorpe, D. Fize, and C. Marlot, "Speed of processing in the human visual system," *nature*, vol. 381, p. 520, Jun. 1996.
- [26] J. Song, R. Lanka, A. Zhao, Y. Yue, and M. Ono, "Learning to search via self-imitation with application to risk-aware planning," in *Proc. Adv. Neural Inform. Process. Syst. Workshop*, Dec. 2017.
- [27] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, 2010.
- [28] M. Chiang, P. Hande, T. Lan, C. W. Tan, *et al.*, "Power control in wireless cellular networks," *Found. Trends Networking*, vol. 2, no. 4, pp. 381–533, 2008.
- [29] C. Michelangelo, C. Gérard P., and Z. Giacomo, *Integer Programming (Graduate texts in mathematics)*, vol. 271. Springer Heidelberg, 2014.
- [30] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, p. 484, Jan. 2016.
- [31] Y. Tian, Q. Gong, W. Shang, Y. Wu, and C. L. Zitnick, "ELF: An extensive, lightweight and flexible research platform for real-time strategy games," in *Proc. Adv. Neural Inform. Process. Syst.*, pp. 2659–2669, Dec. 2017.
- [32] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Found. Trends Mach. Learning*, vol. 11, no. 3-4, 2018.
- [33] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, *et al.*, "An algorithmic perspective on imitation learning," *Found. Trends Robotics*, vol. 7, pp. 1–179, Mar. 2018.
- [34] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. Int. Conf. Artificial Intell. Stat.*, pp. 627–635, Apr. 2011.
- [35] M.-F. Balcan, T. Dick, T. Sandholm, and E. Vitercik, "Learning to branch," in *Proc. Int. Conf. Mach. Learning*, vol. 80, pp. 344–353, Jul. 2018.
- [36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inform. Process. Syst.*, pp. 5998–6008, Dec. 2017.
- [37] A. R. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pp. 3712–3722, Jun. 2018.
- [38] A. Gjendemsjø, D. Gesbert, G. E. Øien, and S. G. Kiani, "Binary power control for sum rate maximization over multiple interfering links," *IEEE Trans. Wireless Commun.*, vol. 7, pp. 3164–3173, Aug. 2008.
- [39] Y. Cheng, M. Pesavento, and A. Philipp, "Joint network optimization and downlink beamforming for CoMP transmissions using mixed integer conic programming," *IEEE Trans. Signal Process.*, vol. 61, pp. 3972–3987, May 2013.