A Machine Learning Based Algorithm for Joint Scheduling and Power Control in Wireless Networks

Xianghui Cao*, Senior Member, IEEE, Rui Ma*, Lu Liu†, Member, IEEE, Hongbao Shi*, Yu Cheng†, Senior Member, IEEE, and Changyin Sun*

Abstract—Wireless network resource allocation is an important issue for designing Internet of Things systems. In this paper, we consider the problem of wireless network capacity optimization that involves issues such as flow allocation, link scheduling and power control. We show that it can be decomposed into a linear program and a nonlinear weighted sum-rate maximization problem for power allocation. Unlike most traditional methods that iteratively search the optimal solutions of the nonlinear subproblem, we propose to directly compute approximated solutions based on machine learning techniques. Specifically, the learning systems consist of both support vector machines (SVMs) and deep belief networks (DBNs) that are trained based on offline computed optimal solutions. In the running phase, the SVMs perform classification for each link to decide whether to use maximal transmit power or be turned off. At the same time, the DBNs compute an approximation of the optimal power allocation. The two results are combined to obtain an approximated solution of the nonlinear program. Simulation results demonstrate the effectiveness of the proposed machine learning based algorithm.

Index Terms—Wireless network, scheduling, power control, joint optimization, deep learning, support vector machine, deep belief network

I. INTRODUCTION

MACHINE learning techniques have been widely applied in various areas such as data mining, computer vision, ranking or recommendation systems, and pattern or abnormality detection [1]–[5]. Recently, machine learning has also found its applications in wireless networks to address complex computation issues raised by the large network size or high big data volumes [6]–[11]. Most of such applications however focus on utilizing machine learning techniques to deal with the data sets themselves as those done in traditional data analysis applications. It is not fully explored what role machine learning can play on those fundamental research issues in wireless networks such as scheduling, resource allocation, and optimization. In this paper, we make an effort to study how to exploit machine learning for solving complex optimal resource allocation issues in wireless networks.

In the literature, machine learning has been exploited from different aspects to address networking issues, with the objectives of enhancing performance, developing new protocols, or reducing computation complexity. For example, deep neural networks have been established in [8] to solve routing problems in dynamic traffic environments. Machine learning techniques also have been applied in MAC protocol design [12], quality-of-service (QoS)-aware resource allocation [13], network traffic classification [14], flow prediction [15] and mobility prediction [16]. Despite the various applications of machine learning in networking, only a few works lay focus on network optimizations in the sense of scheduling and resource allocation. The work in [17], [18] implements machine learning in wireless network optimization, by performing learning based link evaluations to exclude unused links from problem formulation, so that the computational complexity can be reduced without affecting optimal solutions.

In this paper, we consider the classic yet challenging research issue: how to maximize network capacity through link scheduling and power control, under a physical interference model [19], [20]. In order to obtain a joint optimization solution, existing approaches usually introduce high-complexity algorithms or long convergence time due to the NP hardness nature, where many coupling network resources incur high dimensional optimization variables and constraints. To speed up the convergence, people resort to relaxing the original problem, developing polynomial approximation algorithms or distributed algorithms based on game theory, e.g., in [19], [21]–[23]. However, the solutions in these works either sacrifice optimality for fast convergence, or suffer from long convergence time in order to reach an optimum. In this paper, we take an innovative perspective that the capability of machine learning can be exploited to address such challenging network optimization problem, to achieve a good balance of performance and complexity.

Machine learning can be viewed as a sophisticated tool to extract information and knowledge from the training process based on empirical data. Generally, the results of network optimization (i.e., the maximum network capacity and associated optimal resource allocation) are determined by the input of the problem, such as network topology, flow demands and resource constraints. With such a perspective, the process of solving the optimization problem can be viewed as investigating the relationship between network input and output, which is usually so complicated and impossible to be characterized with an explicit function expression. Nevertheless, such a task may be fulfilled with machine learning techniques for their...
capabilities in extracting high-level abstractions from input data and regression of complicated function. The training data can be obtained from computation experiences, which are the formulation and solutions of historical optimization problems. Based on the training data, a learning framework can be built and trained to extract the desired latent information. Then given a new input (yielding a new optimization problem), the learning framework can respond with an output with utilizations of the learned knowledge such as an estimated function mapping from input to output.

In this paper, we formulate the capacity maximization problem as a joint nonlinear program of flow allocation, link scheduling and power control. The problem is then decomposed into a scheduling and flow control subproblem (master problem) which is a linear program and a power control subproblem (slave problem) which is nonlinear. The two subproblems can be solved recursively to achieve the optimum of the original capacity optimization problem. We show that the slave subproblem is equivalent to the weighted sum-rate maximization problem, which reflects the main complexity of solving the original problem. Unlike traditional methods to iteratively search the optimal solutions of the slave subproblem, we propose to apply machine learning techniques to directly approximate the optimal subproblem solution given the input from the solution of the master problem. Specifically, we establish a number of support vector machines (SVMs) and deep belief networks (DBNs) to learn the solution structure of the power control subproblem. The proposed method consists of three phases, namely data preparation, training and running phases. The learning systems are trained based on offline optimal solution samples obtained by off-the-shelf optimization tools. After training and in the running phase, given an input from the master problem, the SVMs are first applied to classify the extreme cases, i.e., whether a link should be assigned the maximum or 0 power. Then, the DBNs are used to obtain fine-grained power allocations. The two results are combined in a probabilistic manner to output the solution to the master problem. The proposed method is evaluated and compared with other methods through simulations.

Our main contributions can summarized as follows: (1) We explore machine learning techniques to address complex resource allocation problems in wireless networks. (2) We decompose the network capacity optimization problem and propose a machine learning based method integrated with SVMs and DBNs to solve the nonlinear part of the optimization problem. (3) Simulation results demonstrate that the proposed learning based method achieves close performance to an optimal iterative algorithm but at a much lower cost.

The remainder of this paper is organized as follows. Section II reviews more related work. Section III presents the network models and formulates the capacity optimization problem. Section IV decomposes the problem. The learning-based algorithm is presented in Section V. Section VI presents the simulation results and Section VII concludes the paper.

II. RELATED WORK

In this paper, we show that the main difficulty in optimally solving the decomposed capacity optimization problem lies in the complexity of the nonlinear subproblem, which is known as the non-convex weighted sum-rate maximization problem. In the literature, this problem and its closely related variant—the max-min rate problem—have been studied, e.g., in [19], [24], [25]. In [21], the sum-rate maximization problem is transformed into a multiplicative linear fractional programming, which can be solved by an iterative algorithm. In [19], Tan et al. apply nonnegative matrix inequalities to transform the original problem with power as the variables to one that directly optimizes the signal-to-noise-plus-interference ratios (SINRs). They show that the problem can further be transformed into a convex optimization and the optimal solutions are achieved at the boundary of a convex set. Instead of directly solving such a problem, two relaxation techniques are used and two iterative algorithms are proposed to achieve near optimal solutions. Similarly, a geometrically fast convergence algorithm with approximately optimal solutions is proposed in [22], and a suboptimal scheduling policy under congestion control is obtained to solve the sum-rate maximization problem. However, such iterative algorithms require a long convergence time until an optimal (or a suboptimal) solution can be achieved. Both exact computational tools with column generation and approximation alternatives are proposed in [24], for maximizing minimum throughput in wireless mesh networks. These algorithms can achieve near optimal solutions with less time, but at the cost of loss in optimality.

The main idea of this work is to apply machine learning techniques to approximate the optimal solutions of the nonlinear subproblem of power control. In the literature, there are some related works on learning-based approaches for various issues of networking. The authors in [8] propose a deep learning based routing algorithm for high-speed core networks where they explore learning capabilities of smart routers to directly compute routing decisions. In this sense, compared to traditional rule-based routing policies, the learning-based method can reduce signaling overhead and the routing decisions can better respond to network traffic dynamics. The proposed learning system in [8] consists of multiple restricted Boltzmann machines (RBMs) with the input of the bottom RBM as the traffic pattern and the output of the top RBM as the routing decision vector. Machine learning based MAC protocols have been investigated in [12] for energy saving, where the energy effectiveness is estimated with Bayesian method under conjugate prior information. For quality of experience provisioning in mobile networks, a dynamic resource allocation scheme is proposed in which various machine learning techniques (e.g., bagging tree, boosted tree, K-nearest neighbor, support vector machine, and Kohonen networks) are applied to predict the network performance based on measurement data from users such as SINR, throughput and delay [13]. The authors show that bagging tree outperforms others in their case. Machine learning is also adopted in areas such as network traffic classification [14], flow prediction [15] and mobility prediction [16], to fulfill traffic analysis and control. A model-free reinforcement learning algorithm is presented in [26] for scheduling and resource allocation schemes in heterogeneous networks with hybrid energy to maximize energy efficiency in the network.
### III. Problem Formulation

We consider a static wireless network described by a graph $G(N, L)$ where $N$ is the set of all network nodes while $L$ represents the set of all (directed) physical links between the nodes. For any link $l \in L$, denote $Tx(l)$ and $Rv(l)$ as the transmitting and receiving nodes of $l$, respectively. Suppose that there are a set $F$ of commodity flows traversing in the network with each flow specified by a pair of source and destination nodes. Each flow has a rate demand of $q_f$ by users. Our objective is to maximize the network capacity to best satisfy user demands while keeping a certain level of fairness. This requires jointly solving the coupled issues of flow allocation, link transmission time scheduling and transmit power control. The routing information can be inferred from the flow allocation and link scheduling results, to be explained later. Hereafter, we assume that each flow has at least one path from its source to its destination; otherwise, the flow demand cannot be achieved. Table I summarizes the main abbreviations and notations used throughout this paper.

#### TABLE I: Abbreviations and Notations

<table>
<thead>
<tr>
<th>Term</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM</td>
<td>Support vector machine</td>
</tr>
<tr>
<td>DBN</td>
<td>Deep belief network</td>
</tr>
<tr>
<td>SINR</td>
<td>Signal-to-noise-plus-interference ratio</td>
</tr>
<tr>
<td>RBM</td>
<td>Restricted Boltzmann machine</td>
</tr>
<tr>
<td>MLA</td>
<td>The proposed machine learning based algorithm</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic algorithm</td>
</tr>
<tr>
<td>$N$</td>
<td>Set of all network nodes</td>
</tr>
<tr>
<td>$L$</td>
<td>Set of all physical links</td>
</tr>
<tr>
<td>$F$</td>
<td>Set of all commodity flows</td>
</tr>
<tr>
<td>$s(f), d(f)$</td>
<td>Source and destination nodes of flow $f$, respectively</td>
</tr>
<tr>
<td>$q_f$</td>
<td>Rate demand of flow $f$ by user</td>
</tr>
<tr>
<td>$p_l$</td>
<td>Transmit power of link $l$</td>
</tr>
<tr>
<td>$\gamma_f, \lambda_f$</td>
<td>The SINR and achieved capacity of link $f$, respectively</td>
</tr>
<tr>
<td>$r_{f,l}$</td>
<td>The rate allocated to link $l$ for flow $f$</td>
</tr>
<tr>
<td>$\theta$</td>
<td>The network capacity under fairness consideration</td>
</tr>
</tbody>
</table>

Throughout this paper, we adopt the following notations: we use lower-case letters in boldface to represent vectors (column vectors by default) and capital letters to denote matrices or constant numbers. All numbers, vectors and matrices in this paper take real values. For a vector $v$, $[v]_i$ is its ith element; similarly, $[M]_{i,j}$ is the $(i,j)$th entry of matrix $M$. For a matrix $M$, $M^t$ is its transpose while $\rho(M)$ is its spectral radius if $M$ is a square matrix. $I$ is the identify matrix whose dimension is clear in the context. For a vector $v$ of dimension $m$, $\text{diag}(v)$ is a $m \times m$ diagonal matrix with ith diagonal entry equals to $[v]_i$. For two vectors $v_1$ and $v_2$ of the same dimension, $v_1 \geq v_2$ represents that $\forall i, [v_1]_i \geq [v_2]_i$. The operator $\leq$ is similarly defined. $|\cdot|$ denotes the number of elements of a set.

#### A. System models

1) Transmit power control: We adopt the following physical interference model to characterize the the relationships among links. For any link $l \in L$, let $p_l \in [0, \bar{p}_l]$ be the transmit power of $Tx(l)$. Denote $p = (p_1, p_2, \ldots, p_{|L|})'$ as a power allocation vector for all the links. Then, $\forall l \in L$, its SINR is given by

$$\gamma_l(p) = \frac{g_{l,l,p_l}}{\sigma_l + \sum_{q \in L \setminus l} g_{l,q} p_q},$$

where $\sigma_l$ is the average noise power on $l$, $\forall l \in L$, $g_{l,l}$ means the channel gain from the transmitter of $l$ to the receiver of $l$, which depends on the path loss, shadowing and fading factors. Particularly, $g_{l,l}$ is the gain of link $l$. A power allocation should avoid the impractical situations that two conflicting links are simultaneously scheduled (or equivalently are allocated positive power) if: (1) they share the same transmitter or (2) the receiver of one link is the transmitter of the other, due to the half-duplex constraint. Hence, at any time, if a node is scheduled to transmit data, at most one of its outbound links is scheduled. To characterize such conflicts, we define a matrix $C$ with each entry as:

$$c_{l,l'} = \begin{cases} 1, & \text{if } \{Tx(l), Rv(l')\} \text{ and } \{Tx(l'), Rv(l)\} \text{ share a common node except that } Rv(l) = Rv(l') \\ 0, & \text{otherwise} \end{cases}$$

and the conflicting constraint of two links can be captured by $c_{l,l'} p_l p_{l'} = 0, \forall l,l' \in L$. (2)

In the following, we say a power allocation $p$ is feasible if the above equation holds.

Given a feasible power allocation $p$, the capacity (maximum achievable rate) of a link $l$ can be calculated as follows:

$$\lambda_l(p) = \begin{cases} B \log(1 + \gamma_l(p)), & \text{if } \gamma_l(p) \geq \gamma_l \\ 0, & \text{otherwise} \end{cases}$$

where $B$ is the channel bandwidth. $\gamma_l$ denotes the SINR threshold of link $l$ below which the received signal cannot be successfully decoded by the receiver.

2) Link schedule: Since $p$ is continuous, there are infinitely many possible power allocations each of which can allow multiple links, if not conflicting, to transmit data concurrently. For the problem tractability and that practically we cannot schedule uncountably many power allocations, we consider a finite number of allocations for ease of presenting the mathematic problem formulation. Let $P = [p_1, p_2, \ldots, p_K]$ be the matrix of a sufficiently large number of feasible power allocations. Then, the scheduling subproblem is to allocate the transmission time for each $p_i$. Consider a finite scheduling time horizon and define $\alpha_i$ as the portion of total time allocated to $p_i$. Denote $\alpha = [\alpha_1, \ldots, \alpha_K]$. We must have

$$\sum_{i=1}^K \alpha_i = 1.$$ (4)

3) Network flow control: For each flow $f$, define a $|N| \times |L|$ incidence matrix $H_f$ where $\forall i \in N$ and $\forall f \in L$.

$$[H_f]_{i,l} = \begin{cases} 1, & \text{if } i = Tx(l) \\ -1, & \text{if } i = Rv(l) \\ 0, & \text{otherwise} \end{cases}$$

Let $r_f = [r_{f,1}, \ldots, r_{f,|L|}]'$ be the flow allocation vector for flow $f$. Then, if the demand of $f$ is exactly satisfied, we have

$$H_f r_f = q_f.$$ (6)
where \( \forall i \in \mathcal{N} \), the \( i \)-th element of vector \( \mathbf{q}_f \) is

\[
[q_f]_i = \begin{cases} 
q_f, & \text{if } i = s_f \\
-q_f, & \text{if } i = d_f \\
0, & \text{otherwise}
\end{cases}
\]

Equation (6) is called the flow balance constraint, which ensures that the import and export rates of a flow are balanced at each non-source and non-destination node. In this paper, considering the fairness among all the flows, we introduce the scaling factor \( \theta \) for network capacity optimization in the sense that the final achieved rate of each flow is \( \theta \) times of its demand, i.e., \( \forall f \in \mathcal{F}, \)

\[
H_f \mathbf{r}_f = \theta \mathbf{q}_f.
\]

Moreover, the flow allocated to each link is upper-bounded by its capacity after scheduling, i.e., \( \forall \ell \in \mathcal{L} \),

\[
\sum_{f \in \mathcal{F}} r_{f,\ell} \leq \sum_{i=1}^{K} \alpha_i \lambda_{e}(\mathbf{p}_i).
\]

According to [27], we can always choose \( \alpha \) such that the left-hand side of (9) exactly meets the right, which will then benefit the network capacity. Therefore, in the following, (9) is treated as an equality.

B. Problem Formulation

With the above models, the joint transmit power control, scheduling and flow allocation problem for capacity optimization under fairness consideration is formulated as follows:

**Problem 1 (Original problem):**

\[
\begin{align*}
\max_{\alpha, \mathbf{r}, \mathbf{P}_k} & \quad \theta \\
\text{s.t.} & \quad (1), (2), (3), (4), (8), (9) \\
& \quad 0 \leq p \leq \bar{p}_\ell, \quad \forall \ell \in \mathcal{L} \\
& \quad \alpha_i \geq 0, \quad i = 1, 2, \ldots \\
& \quad r_{f,\ell} \geq 0, \quad \forall \ell \in \mathcal{L}, \forall f \in \mathcal{F} \\
& \quad (\gamma_{f}(\mathbf{p}_i) - \gamma_{f}) p_i \geq 0, \quad \forall \ell \in \mathcal{L}, i = 1, 2, \ldots
\end{align*}
\]

where the last inequality ensures that either \( p_\ell \) is 0 or both \( p_\ell > 0 \) and \( \gamma_{f} \geq \gamma_{f} \) for each \( \ell \). Notice that the solutions of \( \mathbf{r} \) and \( \alpha \) together tell that when and at what rate a link transmits which flow’s data. At the same time, the flow balance constraint in (8) ensures that a flow from its source will be delivered to its destination. Therefore, for each flow, the scheduled links will connect the source and destination nodes, which indicate the routes the flow traverses.

The above problem is a nonlinear program due to the nonlinearity in (1) and (3). Furthermore, its difficulty also lies in the impossibility of enumerating all the power allocations \( \{\mathbf{p}\} \). In the following, we resort to a problem decomposition technique to develop an effective algorithm.

IV. PROBLEM DECOMPOSITION

The decomposition is based on column generation technique [27]–[29]. Instead of enumerating all the power allocations, we gradually insert new power allocations to existing set of allocations step-by-step. Let \( \mathbf{P}_k \) be the matrix of all feasible power allocations obtained by step \( k \) and \( \mathbf{p}_k \) be the new feasible allocation found in step \( k \) to be added to \( \mathbf{P}_k \). Let \( n_k \) be the number of allocations (columns) in \( \mathbf{P}_k \). Then,

\[
\mathbf{P}_{k+1} = [\mathbf{P}_k, \mathbf{p}_k] \quad \text{and} \quad n_{k+1} = n_k + 1.
\]

We start with a small set of feasible allocations, say \( \mathbf{P}_0 = \{\mathbf{p}_0^1, \ldots, \mathbf{p}_0^n\} \). At any step \( k \) with a given \( \mathbf{P}_k \), Problem 1 reduces to a joint scheduling and flow allocation problem which can be represented in a matrix form as follows:

**Problem 2 (Master subproblem):** The fixed-power joint scheduling and flow allocation problem at step \( k \) is:

\[
\begin{align*}
\max_{\mathbf{x}_k} & \quad \mathbf{u}^T \mathbf{x}_k \\
\text{s.t.} & \quad A_k \mathbf{x}_k = \mathbf{b} \\
& \quad \mathbf{x}_k \geq \mathbf{0}
\end{align*}
\]

where

\[
\begin{align*}
\mathbf{x}_k &= [\theta, \mathbf{r}_k', \alpha_k']^T, \\
\mathbf{u}_k &= \left[1, 0_{|\mathcal{L}|+|\mathcal{F}|}^T\right]', \\
\mathbf{b} &= \left[0_{|\mathcal{N}|+|\mathcal{F}|}^T\right]', \\
A_k &= \begin{bmatrix}
-q_1 & H_1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
-q_{|\mathcal{F}|} & 0 & \cdots & H_{|\mathcal{F}|}
\end{bmatrix} - I_{|\mathcal{F}|} \lambda(\mathbf{P}_k)^T,
\end{align*}
\]

In the above, \( \mathbf{r}_k = [r_1', \ldots, r_{|\mathcal{F}|}']^T \), \( I_f \) is the identity matrix of dimension \( |\mathcal{L}| \times |\mathcal{L}| \), \( 1_n \) is an \( n \)-sized vector with all elements equal to 0. \( 1_n \) is similarly defined. \( \lambda(\mathbf{P}_k) = [\lambda(\mathbf{p}_1), \ldots, \lambda(\mathbf{p}_n)] \) and \( \lambda(\mathbf{p}_i) = [\lambda_1(\mathbf{p}_i), \ldots, \lambda_{|\mathcal{L}|}(\mathbf{p}_i)] \).

Clearly, Problem 2 is a linear program that can be efficiently solved. Starting from the initial allocation set \( \mathbf{P}_0 \), by (22), adding a new \( \mathbf{p}_k \) to \( \mathbf{P}_k \) is equivalent to adding a new column \( \mathbf{a}_k = [0_{|\mathcal{N}|+|\mathcal{F}|}^T, \lambda(\mathbf{p}_k)]' \) to form \( A_{k+1} = [A_k, \mathbf{a}_k] \). Based on the column generation method [28], \( \mathbf{a}_k \) is chosen as the one that maximizes the reduced cost as follows:

\[
-\bar{\mathbf{w}}_k' \mathbf{a}_k = -\bar{\mathbf{w}}_k' \lambda(\mathbf{p}_k) + 1 - \sum_{\ell=1}^{|\mathcal{L}|} \tilde{w}_{k,\ell} \lambda_{e}(\mathbf{p}_k)
\]

where \( \bar{\mathbf{w}}_k \) is the optimal dual associated with the constraint in (22). Let \( \mathbf{w}_k = [-\bar{\mathbf{w}}_k, \lambda(\mathbf{p}_k)]' \), then, we have the following theorem.

**Theorem 1:** To find a new feasible power allocation \( \mathbf{p}_k \) to add to \( \mathbf{P}_k \) is equivalent to solving a weighted sum-rate maximization problem under a conflicting constraint and a minimum SINR constraint as stated in Problem 3.

**Problem 3 (Slave subproblem):**

\[
\begin{align*}
\max_{\mathbf{p}} & \quad \mathbf{w}^T \lambda(\mathbf{p}) \\
\text{s.t.} & \quad 0 \leq \mathbf{p} \leq \bar{\mathbf{p}} \\
& \quad \text{diag}(\mathbf{p})' \text{diag}(\mathbf{p}) = 0 \\
& \quad \text{diag}(\mathbf{p})' (\gamma(\mathbf{p}) - \gamma) \geq 0.
\end{align*}
\]

This paper considers several problems for which the relationships are displayed in Fig. 1. The original optimization
problem (i.e., Problem 1) is decomposed into a master problem (Problem 2) and a slave problem (Problem 3) which is a nonlinear. Problems 2 and 3 are solved recursively. In each iteration, we first solve Problem 2, which outputs \( w_k \) to form Problem 3. Then the latter is solved, which outputs a new power allocation vector \( p_k \) that is then used to update Problem 2. The iterations stop when no new \( p_k \) can be found, i.e., the achieved objective in Problem 3 is non-positive, indicating that an optimal solution to Problem 1 is obtained.

**Remark 1:** Since \( \lambda_\ell(p) \geq 0 \), the optimal power for link \( \ell \) is 0 if \( w_\ell \leq 0 \). Therefore, for those \( w_\ell \leq 0 \), we can simply set them 0, which will not change the optimal solution of Problem 3. Meanwhile, the optimal solution also remains if the weights in \( w_k \) are normalized by dividing their maximum value. Thus, in the following, it is without loss of generality to assume that \( w_\ell \in [0, 1] \) for all \( \ell \) in the above problem.

The initial power allocation matrix \( P_0 \) can be arbitrarily given as long as all \( \{p_0^\ell\} \) are feasible (i.e., no conflicting links are scheduled at the same time) and cover all links (i.e., each link is scheduled in at least one of \( \{p_0^\ell\} \)). Therefore, a simple method is to set \( P_0 = [\bar{p}_1 e_1, \ldots, \bar{p}_r e_r] \) with \( n_0 = |L| \), where \( e_i \) is a \( |L| \)-sized vector with \( i \)-th element equals to 1 and all the others equal to 0.

Problem 3 is a non-convex nonlinear program which is still difficult to solve. In the literature, there are some iterative algorithms developed to solve this sum-rate maximization problem effectively [19], [25]. However, such algorithms usually require a long time to converge. In the following section, we propose a learning-based method to directly provide an approximated solution of Problem 3.

### V. THE LEARNING-BASED ALGORITHM

An overview of the proposed learning-based method is illustrated in Fig. 2. The main idea is to establish a set of support vector machines and neural networks to approximate the optimal solutions of Problem 3 by offline learning a large set of optimal solution samples. The method consists of three phases. In the first data preparation phase, we apply some off-the-shelf nonlinear programming algorithm to solve Problem 3 and obtain a sufficient number of samples. Each sample is denoted as a pair \( (w, y) \), where \( y \) is a function of the optimal solution given \( w \). In the second phase, a set of SVMs and DBNs are established and trained by the obtained samples \( \{(w, y)\} \), where the input of each SVM and DBN is \( w \) while the output is \( \{0, 1\} \) for SVM and \( y \leq \bar{y} \) for DBN. The SVMs and DBNs after training will be used to approximate the solutions of Problem 3 online in the third phase. The whole learning-based algorithm (MLA) that iteratively solves the original problem is summarized in Algorithm 1.

### A. Preliminaries

Before presenting the detailed algorithm design, we first briefly introduce the SVM and DBN. SVM is a supervised learning method normally for classification, i.e., to decide the category a new data point best belongs to [30]. Consider the linear classification case. For a set of training data that can be viewed as points in some space, given that each point is labeled by either one of two categories, two subsets of the points are formed. SVM is to find the maximum-margin hyperplane that separates the two subsets with the largest separation. In other words, training a linear SVM is to determine a hyperplane that maximizes the distances from it to the two subsets. Detailed designs are presented in Section V-C1.

DBN is an effective class of deep learning networks [31]. As shown in Fig. 3, an example of the structure of DBN consists of an input layer, a number of hidden layers and an output layer, which is also a stack of RBMs. Each RBM is an undirected, generative energy-based model consisting of a visible layer and a hidden layer. As shown in this figure, the bottom RBM takes the input layer as its visible layer. The next RBM takes the hidden layer of the previous RBM as its visible layer and the next hidden layer as its own hidden layer. The neurons in different layers are connected with each connection associated with a weight, while those belonging to the same layer are disconnected. The training process for each DBN contains two steps. In the first step, the RBMs are trained layer-by-layer with unsupervised learning, while in the second step the whole DBN is fine-tuned with supervised learning based on the backpropagation method. Detailed DBN design and training process are presented in Section V-C2.

### B. Data preparation phase to obtain training sets

We need to obtain a sufficient number of sample optimization solutions in order to train the machine learning systems. Notice that the objective function in Problem 3 is nonlinear and contains non-smooth points (see (3)). Since \( \Sigma_{\ell} \) usually is small, we relax Problem 3 by ignoring \( \{z_\ell\} \) as follows:

**Problem 4 (Relaxed slave problem):**

\[
\begin{align*}
\max_p & \quad \sum_{\ell \in L} w_{k,\ell} \log(1 + \gamma_\ell(p)) \\
\text{s.t.} & \quad 0 \leq p \leq \bar{p} \\
& \quad \text{diag}(p)C\text{diag}(p) = 0.
\end{align*}
\]

The optimal solutions to the above problem can be searched by some optimization algorithms such as genetic algorithms and simulated annealing. In our simulations, the genetic algorithm is adopted to generated sample solutions. As shown in Fig. 4, from a typical distribution of the optimal power solutions, we can observe that the optimal power of most links are either within a small range of 0 or close to their maximal value \( \{\bar{p}_k\} \).
This is reasonable since in the optimal solution, interference is minimized in order to improve the link rates, likely resulting in either small or large power allocations. Besides, since the threshold $\gamma_\ell$ is ignored in Problem 4, the optimal solution may yield very small $p_\ell^*$’s. In view of this, we apply SVMs to classify the power allocation for each link into three categories, i.e., close to 0, close to its maximal transmit power and away from the two extreme values, in order to provide a first-step rough approximation of the optimal power allocation. In this case, the samples of input weight and optimal solution pairs (i.e., $\{(w,y_{\text{SVM}})\}$ with $y_{\text{SVM}} = p^*$) are used to train the SVMs, where $p^*$ is the optimal solution of Problem 4.

In order to refine the optimal power approximation, instead of dwelling on $\{(w,p^*)\}$, we directly use the link SINRs to train the neural networks. We apply deep learning here in order to take the potential advantage of its superior performance in many fields such as signal processing, computer vision, bioinformatics and so on. DBN is a typical and one of the first effective deep learning algorithms. In this paper, we apply DBN to learn the data of link weights (input) and SINRs (output). First, we transform the above problem into a new problem that directly optimizes the SINRs. Define the following vectors and matrix:

$$g = [g_{1,1}, \ldots, g_{|\mathcal{L}|,|\mathcal{L}|}]^T$$

$$\sigma = \begin{bmatrix} \sigma_1, & \ldots, & \sigma_{|\mathcal{L}|}\end{bmatrix}^T$$

$$G = \begin{bmatrix} g_{1,1} & \cdots & g_{1,|\mathcal{L}|} \\ \vdots & \ddots & \vdots \\ g_{|\mathcal{L}|,1} & \cdots & g_{|\mathcal{L}|,|\mathcal{L}|} \end{bmatrix} - I$$

where $\mathcal{L}$ is the set of links.
Then, the relationship between \( p \) and \( \gamma \) if \( \{ \gamma_i \} \) are ignored can be described in a matrix form as follows.

\[
(I - \text{diag}(\gamma)G)p = \text{diag}(\gamma)\tilde{\sigma}
\]

Based on the above equation and according to [19], Problem 4 can be equivalently transformed into:

**Problem 5 (Transformed relaxed slave problem):**

\[
\begin{align*}
\max_{\Gamma = \text{diag}(\gamma)} & \quad \mathbf{1}^t \log \left( 1 + \Gamma \right) w \\
\text{s.t.} & \quad \rho \left( \Gamma \tilde{G}_t \right) \leq 1, \quad \forall t \in \mathcal{L} \\
& \quad \Gamma \mathbf{C}_t = 0.
\end{align*}
\]

where \( \tilde{G}_t = G + \frac{1}{p_t} \varepsilon e_t \). Notice that the constraint (34) is equivalent to (26) since (2) is equivalent to \( c_t \gamma_t \ell_t = 0 \).

The objective function in the above problem is strictly concave in which the variables \( \{ \gamma_i \} \) are decoupled. We define \( y_{\text{DBN}} = \lambda(\gamma^*) \) and use \( \{(w, y_{\text{DBN}}) \} \) as the training set to train the neural networks, where \( \gamma^* \) is the optimal solution of Problem 5. Accordingly, the input and output of the neural networks become \((w, \lambda(\gamma^*))\).

To obtain the training sets \( \{(w, y)\} \), each time the input \( w \) is uniformly randomly generated within \([0, 1]\) with some of the elements set as 1. Our simulations suggest that in many cases solving Problem 2 in the running phase will generate sparse \( w \)'s, i.e., some elements are 0. However, we tried not to use sparse \( w \)'s to compute the training set due to the reduction of information the neural networks can learn. In fact, suppose \( w^t = 0 \), the optimal \( p^t_\ell \) is 0 if \( w \neq 0 \). In addition, if \( w = 0 \), any \( \mathbf{p}^* \) will be the optimal solution, which is another reason to avoid using too sparse \( w \)'s.

**C. Training phase**

In this phase, we train two SVM classifiers and a DBN for each link separately.

1) The SVM classifiers: Let \( \omega_0 \), \( \omega_1 \) and \( \omega_2 \) represent the classes of \( \omega_0 = \{ y : y \leq \delta_{t,1} \} \), \( \omega_1 = \{ y : \delta_{t,1} \leq y \leq \delta_{t,2} \} \) and \( \omega_2 = \{ y : y \notin \omega_0, y \notin \omega_1 \} \), respectively, where \( \delta_{t,1} \) and \( \delta_{t,2} \) are two small threshold numbers. To classify a sample, two SVMs are established and trained. The first SVM decides whether a sample belongs to \( \omega_0 \) or not. Hence, we label the samples as \( z^\ell = 1 \) if \( y^\ell \in \omega_0 \) and \( z^\ell = -1 \) otherwise. The output of the SVM can be written as follows:

\[
z^\ell_0 = \text{sign} \left( \sum_{i} a'_{0,i} \mathcal{K}(w_i, w) + b_0' \right),
\]

where \( i \) is the index of training samples, \( a'_{0,i} \) is a weight, \( w_i \) is the input of sample \( i \), \( \mathcal{K}(\cdot, \cdot) \) is the kernel function and \( b_0' \) is the bias. In this paper, the kernel functions are chosen linear for simplicity. Then, \( y_i \) corresponding to \( w_i \) is classified to \( \omega_0 \) if \( z^\ell_0 = 1 \), and not to \( \omega_0 \) if \( z^\ell_0 = -1 \). In the special case that \( z^\ell_0 = 0 \), \( y_i \) can either belong to \( \omega_0 \) or not. Thus, the training process for finding the maximum-margin hyperplane is equivalent to solving the following minimization problem:

\[
\min \| a'_{0,i} \|
\quad \text{s.t.} \quad z^\ell_0(a'_{0,i} w + b_0') \geq 1 - \zeta_i, \quad \forall i
\]

where \( \| \cdot \| \) represents the norm of a vector. \( a'_{0,i} \) is the vector formed by \( \{ a'_{0,i} \} \), \( \zeta_i \geq 0 \) is a constant to tolerate classification faults. The SVM for deciding whether a sample lies in \( \omega_2 \) or not is similarly trained.

2) The DBNs: We establish for each link a DBN, denoted as \( DBN^\ell \), to provide a refined approximate of the optimal power allocation, as shown in Fig. 3. For each RBM, let \( \mathbf{v} \) and \( \mathbf{h} \) denote the vectors of the visible and hidden neurons, respectively. The unsupervised training in the first step is to initialize the parameters, including the weights (denoted as matrix \( \Phi \)) between the two visible and hidden layers, the biases (denoted as vectors \( b_v \) and \( b_h \)) associated with the visible and hidden neurons, respectively, at an optimal configuration. Let the parameters be \( \varphi = (\Phi, b_v, b_h) \). Then, the parameters are updated iteratively as follows:

\[
\varphi_{t+1} = \varphi_t + \eta \frac{\partial \log \mathbb{P}(\mathbf{v}_t)}{\partial \varphi},
\]

where \( \eta \) is the learning rate. \( \mathbb{P}(\mathbf{v}) \) is the probability of \( \mathbf{v} \) (one of the training data), which can be given as

\[
\mathbb{P}(\mathbf{v}) = \sum_{h} \mathbb{P}(\mathbf{v}, h) = \sum_{h} \sum_{v} \sum_{h} \exp(-E(\mathbf{v}, h)).
\]

In the above, \( \mathbb{P}(\mathbf{v}, h) \) is the joint distribution of the visible and hidden layers. \( E(\mathbf{v}, h) \) represents the energy function associated with the RBM and takes the form

\[
E(\mathbf{v}, h) = -\mathbf{v}^t \Phi \mathbf{h} - b_v \mathbf{v} - b_h \mathbf{h}.
\]

Notice that it is of great computation complexity to calculate \( \mathbb{P}(\mathbf{v}, h) \) as in (37). To this end, the contrastive divergence method as proposed in [32], which approximates \( \mathbb{P}(\mathbf{v}, h) \) based on Gibbs sampling, may be applied.

The supervised training process is to fine tune the parameters with the output \( y^\ell_\ell \) of sample data by minimizing a cost function called the cross entropy:

\[
S = -\frac{1}{D} \sum_{i=1}^{D} \left( y^\ell_i \log(\hat{y}^\ell_i) + (1 - y^\ell_i) \log(1 - \hat{y}^\ell_i) \right),
\]

where \( y^\ell_i \) and \( \hat{y}^\ell_i \) are the output of the \( i \)th training data and that of the predicted output by the \( DBN^\ell \) given the input of the \( i \)th training data, respectively. In fact, \( S \) is a measure of the prediction accuracy of \( DBN^\ell \). Then, by the backpropagation method, the parameters are tuned iteratively based on the gradient method as follows:

\[
\varphi_{t+1} = \varphi_t - \tilde{\eta} \frac{\partial S}{\partial \varphi},
\]

where \( \tilde{\eta} \) is the learning rate in the gradient method.

**D. Running phase**

In the running phase, given an input \( w \) (after normalization) from the solution to the master problem, the SVMs are first applied to perform classification for each link. Specifically, for each link \( \ell \), if it belongs to class \( \omega_0 \) (resp. \( \omega_2 \)), its power
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2853661, IEEE Internet of Things Journal

Here, we assume that $\sum_p \xi p$ in (23) higher than some small value.

In this section, we present simulation evaluations of the proposed method. As shown in Fig. 5, we consider a 16-node wireless network deployed in a 800m x 600m square area. The maximal communication range is set as 200m, resulting in 26 undirected links (or totally 52 directed links). The interference range is 250m. The channel gain $q_{\ell,\ell'}$ is set inversely proportional to the square distance between Tx$(\ell')$ and Rx$(\ell)$. The maximum transmit power of each link is $p_\ell$ = 1 unit. The bandwidth of the links is set as $B = 1$. All the nodes can serve as relays for forwarding data until their destinations. The simulation program is built within MATLAB, where we use Deep Neural Network codes [33] and LibLinear library [34] for constructing, training and applying the DBNs and SVMs in our algorithm, respectively. The simulations are conducted on a Dell PowerEdge T630 computer with two Intel E5-2620 CPUs and 16GB RAM.

Each $DBN_\ell$ contains four layers with the input, two hidden and the output layers respectively contain 52, 60, 5 and 1 neurons. We generate random $w$'s and solve the slave problem op-

Algorithm 1: The machine learning based algorithm (MLA) for power control

```
input : Flow demands $\{q_1, \ldots, q_{|L|}\}$
initialize: Initial allocation set $P$
while true do
  solve the linear program: Problem 2;
  obtain the (normalized) weights $w$;
  for $\ell \in L$ do
    use SVM$_\ell$ to do classification;
    use DBN$_\ell$ to predict $y_\ell$;
  end
  calculate $p$ based on (41);
  for $\ell \in L$ do
    if $\omega_0$ then
      $p_\ell$ = 0 w.p. $\phi_0,\ell$;
    else if $\omega_2$ then
      $p_\ell$ = $p_\ell$ w.p. $\phi_2,\ell$;
    end
  end
  calculate $\lambda(p)$;
  if $\lambda_\ell < \gamma_\ell$ then
    $p_\ell$ = 0;
  end
  //address the conflicting links issue
  $L_\ell \leftarrow \{ \ell : p_\ell > 0 \}$;
  while $L_\ell \neq \emptyset$ do
    $l \leftarrow \arg \max \lambda_\ell w_l$;
    for $i \in L_\ell$ and $i \neq l$ do
      if $c_{i,l} \neq 0$ then
        $p_i$ = 0; $L_\ell \leftarrow L_\ell \setminus \{i\}$;
      end
    end
    $L_\ell \leftarrow L_\ell \setminus \{l\}$;
  end
  if $w^\top \lambda(p) < \epsilon$ then
    algorithm stops;
  end
  add column $[0_{|L|,|F|}, \lambda(p), 1]^\top$ to the right side of $P$;
end
```
timely using genetic algorithm to generate sample solutions. 20000 samples are used to train the learning systems. In the running phase, we compare the performance of our proposed algorithm (MLA) with that of three other algorithms: the genetic algorithm (GA) which iteratively searches the optimal solutions, the greedy algorithm (Greedy) which greedily solves the slave problem [27] and a random algorithm (Random), when they are applied to solve the slave problem. Random randomly outputs an allocation vector \( \mathbf{p} \) regardless of the input \( \mathbf{w} \) (conflicting links are pruned by the greedy algorithm similar to that in Algorithm 1).

Fig. 6 shows the performance comparison results among the four algorithms, where we consider two scenarios, namely with and without conflict. For the scenario without conflict, we consider a reduced network with 5 (directed) links selected from the original network and these links that are mutually conflict free, i.e., \( c_{l,l'} = 0 \) for any pair of the links. Such a reduced network is disconnected; however, the different parts of the network may have mutual interference. Five source-destination pairs (i.e., \((1, 2), (9, 10), (6, 7), (4, 5)\) and \((14, 10)\)) are chosen with the flow demands as \(2, 0.2, 1.5, 1.5\) and \(0.5\), respectively, which corresponds to Case 4 in Table II. In this case, each \( DBN_{c} \) contains four layers with the input, two hidden and the output layers respectively contain 5, 20, 4 and 1 neurons. The simulation results in Fig. 6(a) show that the proposed MLA achieves very close performance to the GA, indicating its near-optimal performance. Besides, MLA obviously outperforms Greedy in terms of achieved capacity. For the scenario with conflict, we consider the original 52-link network and focus on source-destination pairs \((5, 3), (8, 14)\) and \((6, 15)\) with the corresponding demands \(0.4, 0.5\) and \(1.5\), respectively, which corresponds to Case 3 in Table II. Fig. 6(b) shows similar comparison results as Fig. 6(a) except that GA converges slower than MLA since new power allocations are continuously found. Greedy converges quicker than MLA roughly because that greedily solving the slave problem will activate only a few links, quickly leading to the situation that no new power allocations can be found. In both figures, the Random algorithm achieves the worst performance since it finds new power allocations planlessly.

We further conduct simulations under four cases of network settings as shown in Table II. These cases have different levels of conflicts among the links. For each case, we conduct 50 independent simulation runs, and display in Fig. 7(a) the mean achieved capacity with 95% confidence intervals, where for each run the capacity is calculated as that achieved after 100 iterations. We can observe that MLA always achieves close performance to GA, and obviously outperforms Greedy and Random in all the four cases. In each case, the performance of Greedy almost does not change in different simulation runs; however the performance of all the other three varies because they incorporate probabilistic operations. Particularly, MLA probabilistically accepts the classification results of SVMs, offering more opportunity to finding power allocations. Although MLA is inferior over GA in terms of achieved capacity, its computation time is much lower than that of GA as shown in Fig. 7(b). The two figures also tell that, at similar computing costs, MLA achieves obviously higher capacity than Greedy.

VII. CONCLUSION

We have studied the problem of joint flow allocation, link scheduling and power control for capacity optimization in wireless networks. We decomposed the original problem into a linear program and a nonlinear one, and proposed a machine
TABLE II: Four simulation cases

<table>
<thead>
<tr>
<th>Case</th>
<th># Flows</th>
<th>Source-destination pairs ((\sigma_f, d_f))</th>
<th>Flow demand ((q_f))</th>
<th>Link conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>{(0, 13), (5, 15)}</td>
<td>{0.5, 1.3}</td>
<td>low</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>{(11, 3), (1, 12), (16, 9)}</td>
<td>{1, 1}</td>
<td>moderate</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>{(5, 3), (8, 14), (6, 15)}</td>
<td>{0.4, 0.5, 1.5}</td>
<td>high</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>{(1, 2), (9, 10), (6, 7), (4, 5), (14, 10)}</td>
<td>{2, 0.2, 1.5, 1.5, 0.5}</td>
<td>free</td>
</tr>
</tbody>
</table>

In future work, we will delve into the learning algorithms and similar to that of the simple greedy algorithm. This paper demonstrates the potential advantages of machine learning in wireless sensor networks: algorithms, strategies, and applications, “IEEE Commun. Surveys Tuts.”, vol. 16, no. 2, pp. 1996–2018, 2014.


[23] Y. Wu and D. H. Tsang, “Distributed power allocation algorithm for
spectrum sharing cognitive radio networks with QoS guarantee,” in Proc.

networks: joint scheduling, routing, power control, and rate adaptation,”


[26] Y. Wei, F. R. Yu, M. Song, and Z. Han, “User scheduling and resource
allocation in HetNets with hybrid energy supply: an actor-critic rein-

study of the delayed column generation method for optimizing wireless


efficient capacity optimization in wireless networks,” in Proc. IEEE


5947, 2009.

[32] ——, “Training products of experts by minimizing contrastive diver-

mann machine,” in Proc. IEEE International Conference on Pattern
Recognition (ICPR), 2014, pp. 1526–1531.

[34] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin,
“LibLinear: a library for large linear classification,” Journal of machine