Neural Combinatorial Optimization for Energy-Efficient Offloading in Mobile Edge Computing

QINGMIAO JIANG,† YUAN ZHANG,‡,§ AND JINYAO YAN †,‡,§

†School of Information and Communication Engineering, Communication University of China, Beijing 100024, China
‡State Key Laboratory of Media Convergence and Communication, Communication University of China, Beijing 100024, China
§Key Laboratory of Media Audio and Video, Communication University of China, Beijing 100024, China

Corresponding author: Jinyao Yan (jyan@cuc.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61971382 and Grant 61631016, and in part by the Fundamental Research Funds for the Central Universities.

ABSTRACT Computation offloading is an efficient approach to reduce the energy consumption of a mobile device (MD). In this paper, we consider the multi-user offloading problem for mobile edge computing (MEC) in a multi-server environment. Its aim is to minimize the total energy consumption of MDs. This problem has been proven to be NP-hard. We formulate the problem as a multidimensional multiple knapsack (MMKP) problem with constraints, and propose a neural network architecture called Multi-Pointer networks (Mptr-Net) to solve the problem. We train Mptr-Net based on the reinforcement learning method, and design an algorithm to search for feasible solutions that meet the constraints. The simulation results show that the probability of a Mptr-Net obtaining an optimal solution can exceed 98%, which is approximately 25% more than that of a baseline heuristic algorithm. Additionally, the time needed to solve the problem by our neural network is stable compared with that of a mathematical programming solver named or-tools.

INDEX TERMS Mobile edge computing, multidimensional multiple knapsack problem, multi-pointer networks, reinforcement learning.

I. INTRODUCTION
As one of the key technologies of 5G networks, MEC can effectively augment the computation and energy capacity of MDs by offloading tasks to edge servers (ESs) [1]–[3]. An important problem affecting the computation offloading performance entails request scheduling or decision making [4]. It also is a challenging problem as a whole. In fact, the MEC offloading problem in the multi-server multi-user scenario is NP-hard [5], [6]. Approaches to solving the MEC offloading problem can usually be classified into three categories. The first consists of solver-based solutions using solvers such as SCIP, GLPK, and gurobi. Algorithms of this type are accurate and can produce optimal solutions. Their shortcoming is that they usually require significant and unstable amounts of time to solve the problem. The second category consists of heuristic algorithms that can identify feasible solutions quickly. However, such algorithms cannot provided a performance guarantee. The third category consists of approximation algorithms that can provide theoretically bounded results in polynomial time. Nevertheless, they are usually suitable to particular problem formulations and thus not generally applicable. Therefore, obtaining a solution to the NP-hard MEC offloading problem quickly, automatically and effectively has become a challenge.

With the notable success of deep learning approaches, researchers have begun to construct deep neural networks to solve combinatorial optimization problems [7]–[9]. Enumerating the input and output of an optimization problem and building it into a data set for training the neural network enables the latter to quickly predict output based on a specific input. When exact algorithm-based solvers are used to solve problems, the necessary execution time is rarely stable even for the same input scale. For example, for the NP-hard problem of a traveling salesman, the same input scale, and using the same solver, we observed that the computation time could differ by an order of magnitude. In contrast, the time needed to obtain a response of the trained deep neural network in a
test stage is fixed, and is mainly determined by sampling time. In case of a heuristic algorithm considered as a comparison, exact rules must be manually set by the algorithm’s designer. In contrast, a neural network can automatically learn heuristic rules from the training data, and has significant potential for finding the optimal solution of the problem. Compared with an approximate algorithm, a neural network has a great chance of finding the optimal solution of the MEC offloading problem, while the approximate algorithm can only obtain an approximate solution.

Motivated by the potential ability of neural network to solve combinatorial optimization problems, we introduce neural networks to solving a complex MEC offloading problem. In this paper, we consider a MEC scenario and investigate an energy minimization problem. The network model consists of multiple mobile devices, and multiple servers in a certain cell. There is a task on each MD that can be run locally or offloaded to an ES. We aim to minimize the total energy consumption of MDs during the execution of these tasks. Our main contributions in this paper are as follows:

- We model the MEC offloading problem as a multidimensional multiple knapsack problem (MMKP) [10] with constraints. Each computation task in an MD is regarded as an item that will consume various resources. The ESs are treated as knapsacks with two-dimensional communication and computation resources. In addition, we impose the constraint that all tasks be completed within a certain latency.

- In contrast to the traditional method, we propose a neural network approach to optimize the NP-hard MEC offloading problem. Our work is mainly inspired by Pointer Networks. Unlike a typical knapsack problem that can be solved by Pointer Networks, the MMKP problem represented by a one-dimensional structure has a very large output scale that cannot be directly implemented by Pointer Networks. We present a Mptr-Net architecture that uses a multidimensional pointer to represent the output, and successfully solves the MMKP problem.

- We train the Mptr-Net with an unsupervised reinforcement learning algorithm. The proposed training algorithm is dedicated to training neural networks to learn heuristic rules for solving the MMKP problem. Furthermore, we present an algorithm for searching for feasible solutions to meet the specified latency constraint of every task.

The rest of the paper is organized into the following sections: Section 2 discusses the related research. The system model of our mobile computation offloading problem and the problem formulation are presented in Section 3. Section 4 introduces our approach to solving the problem, including the architecture of Mptr-Net for the MMKP problem, the training process and the search algorithm. The details of simulation and the neural network are shown in Section 5. We then subsequently discuss the results of a performance comparison of our neural network and the or-tools solver. Section 6 concludes the paper.

II. RELATED WORK

Many existing studies consider the MEC offloading problem for energy optimization. Reference [5] design a distributed computation offloading algorithm that can achieve a Nash equilibrium, and extend the analysis to the scenario of multiuser computation offloading in the multi-channel wireless contention environment. Reference [11] use the multi-antenna non-orthogonal multiple access (NOMA) technique for multiuser computation offloading, and develop two low-complexity algorithms to find suboptimal solutions in practice. Reference [12] formulate the offloading problem as a mixed integer nonlinear programming (MINLP) problem, and present a combinational algorithm for obtaining a suboptimal solution. Reference [13] consider the computation offloading problem for multiple access MEC-enabled small cell networks, and design a suboptimal algorithm based on a genetic algorithm (GA). Reference [14] propose an iterative algorithm, based on a novel successive convex approximation technique, converging to a locally optimal solution. Reference [15] develop an artificial fish swarm algorithm-based scheme to solve the energy optimization problem. Unlike our research, the above approaches are mainly based on an approximate algorithm and a heuristic algorithm. It is sometimes necessary to convert the offloading problem into subproblems through relaxation.

Numerous recent studies apply deep learning to mobile computation offloading [16]. Reference [17] propose a technique based on a deep Q-network (DQN) for task migration in a MEC system. It can learn the optimal task migration policy from previous experiences without necessarily acquiring the information about users’ mobility pattern in advance. Reference [18] propose an offloading scheme based on reinforcement learning for an IoT device with energy harvesting to select the edge device and the offloading rate according to the current battery level. Reference [19] use the actor-critic reinforcement learning framework to solve the joint decision-making problem with the objective of minimizing the average end-to-end delay. Reference [20] study the mobile user’s policy to minimize the user’s monetary cost and energy consumption without a known mobility pattern and use a deep Q-network (DQN) for the user to learn the optimal offloading policy from past experiences. These studies have advanced the applications of neural networks in mobile computation offloading, but are different from the scenario and the offloading problem in this article.

Similarly to our scenario, a MEC system is considered by [21] such that every MD has multiple tasks being offloaded to an ES, and the objective is to design a joint task offloading decision and bandwidth allocation. This approach has been extended to multi-server case in [22]. However, the cited studies mainly use relatively simple deep neural networks (DNN) to learn the optimal offloading policy, while we propose a neural network architecture called Mptr-Net to directly
predict the solutions of the MEC offloading problem. In a setting similar to our neural network architecture, [23] address the challenges of task dependency and adapting to dynamic scenarios, and propose a new DRL-based offloading framework that can efficiently learn the offloading policy uniquely represented by a specially designed sequence-to-sequence [29] neural network. Nevertheless, the above study’s network architecture depends on the original Pointer Networks.

III. SYSTEM MODEL AND PROBLEM FORMULATION

A. SYSTEM MODEL

We assume that ESs have relatively stronger computing and storage capability. Mobile users can offload their computation-intensive tasks to ESs over the wireless network. Fig.1 shows our system model. This model is similar to that in many literatures, such as [27] [28]. We consider M ESs that are close to a base station in certain cell, and every MD has one candidate offloading task.

Denote a set of MDs by \( N = \{1, 2, \ldots, N\} \). Each MD has a computation task \( \mathcal{J}_i = (D_i, C_i, T_i^{\text{max}}) \) that is selected as a candidate offloading task through the decision-making algorithm on MD \( j \). \( D_i \) denotes the sum of data size uploaded in connection with task \( \mathcal{J}_i \) and the downloaded result. \( C_i \) denotes the total number of CPU cycles required to complete the computation task \( \mathcal{J}_i \). \( T_i^{\text{max}} \) denotes the maximum tolerable delay in executing task \( \mathcal{J}_i \). Let \( M = \{1, 2, \ldots, M\} \) be the set of possible ESs. Each MD can execute its task locally or remotely. Variable \( \alpha = [\alpha_1, \alpha_2, \ldots, \alpha_M] \) denotes the offloading vector, where \( \alpha_{ij} = 0 \) if MD \( i \) executes the task by the locally, while \( \alpha_{ij} = 1 \) if MD \( i \) offloads the task to ES \( j \) for remote execution.

1) LOCAL EXECUTION MODEL

When a task is executed locally, we denote by \( f_i^L \) the CPU frequency of MD \( i \) assigned to compute task \( \mathcal{J}_i \). \( E_i^L \), the energy consumption of MD \( i \) associated with compute task \( \mathcal{J}_i \), grows with CPU frequency and the number of CPU cycles. Hence, the energy consumption of MD \( i \) is

\[
E_i^L = \kappa (f_i^L)^2 C_i
\]

where \( \kappa \) is the effective switched capacitance [24]. We denote by \( t_i^L \) the execution time, and it is easy to note that

\[
t_i^L = \frac{C_i}{f_i^L}
\]

2) REMOTE EXECUTION MODEL

Tasks \( \mathcal{J}_i \) can be executed remotely (i.e, \( \alpha_{ij} = 1, j \in 1, 2, \ldots, M \)). First, let \( r_i^R \) be the network bandwidth between MD \( i \) and ES \( j \). We assume that the requested network bandwidth does not depend on server choice, and it can be assumed that \( r_i^R = r_j^R, j \in 1, 2, \ldots, M \). Second, let \( f_j^R \) be the frequency of the CPU assigned to compute task \( \mathcal{J}_j \) on ES \( j \). We assume that the number of CPU cycles of each task also does not depend on server choice, and it can be assumed that \( f_j^R = f_i^R, j \in 1, 2, \ldots, M \). We denote by \( P_i^{\text{transfer}} \) the average power of MD \( i \) to upload data to an ES and to download result from ES; \( P_i^{\text{wait}} \) is the power consumption of MD \( i \) in while waiting for the result. If task \( \mathcal{J}_i \) is offloaded to an ES, it should be noted that the total time contains two parts: transfer time \( t_i^{\text{transfer}} \) and execution time \( t_i^{\text{execution}} \).

\[
t_i^{\text{transfer}} = \frac{D_i}{r_i^R} \quad \text{and} \quad t_i^{\text{execution}} = C_i ^{f_i^R}
\]

Let \( E_i^{\text{off}} \) be the energy consumption of MD \( i \) needed to offload task \( \mathcal{J}_i \) to an ES; it can be expressed as

\[
E_i^{\text{off}} = P_i^{\text{transfer}} \times t_i^{\text{transfer}} + P_i^{\text{wait}} \times t_i^{\text{execution}}
\]

B. PROBLEM FORMULATION

Our objective is to minimize the total energy consumption of MDs under the constraints of maximum tolerable delay of tasks, transmission rate and computation capacity. The key symbols used are listed in Table 1. To this end, the problem can be formulated as shown in (5), using the following notation.

\( F_j \) denotes the maximum CPU frequency of ES \( j \), and \( B_j \) denotes the maximum transmission rate of ES \( j \).

C1 is the constraint that the total allocated computational resources of ES \( j \) cannot exceed the maximum frequency \( F_j \).

C2 is the constraint that the sum of allocated transmission rates of all tasks offloaded to ES \( j \) is no larger than the maximum transmission rate \( B_j \).

C3 is the constraint that every task’s delay cannot exceed its maximum tolerable delay.

C4 indicates that any one task will be either offloaded to one ES, or executed locally.

C5 is the constraint on binary variable \( \alpha \), that indicates that each MD executes its task locally or remotely.

\[
\mathcal{P}_{\text{min}} = \min_{\alpha} \sum_{i=1}^{N} \sum_{j=1}^{M} \left[ \alpha_{ij} \times E_i^{\text{off}} + (1 - \alpha_{ij}) \times E_i^L \right]
\]
This problem is formulated as a 0-1 integer programming problem. It is an extension of the knapsack problem that can be proven to be NP-hard [25].

IV. PROBLEM SOLUTION

The problem can be solved directly by the branch-and-bound algorithm. We can use the optimization solver to relax the 0-1 integer programming problem to a linear programming problem, and subsequently use the branching variable to decompose the problem into a series of linear programming subproblems. The solver removes the subproblems without feasible solutions, and takes the maximum lower bound found by a subproblem to be the optimal solution. Because the problem is NP-hard, for a large number of MDs or ESs the computation time needed by a solver will generally be long.

We can also use the constraint programming solver to find the optimal solution to the problem. The latter solver mainly uses heuristic rules.

In addition, we observe that (7) can be divided into 2 parts as follows:

\[
E_{\text{opt}} = \max_{\alpha} \sum_{i=1}^{N} \sum_{j=1}^{M} [\alpha_{ij} \times (E_{i}^{L} - E_{i}^{off})]
\]

s.t. C1, C2, C4, C5

\[
\sum_{i=1}^{M} \sum_{j=1}^{N} \alpha_{ij} \leq N \sum_{i=1}^{N} \sum_{j=1}^{M} [\alpha_{ij} \times (E_{i}^{L} - E_{i}^{off})] - \sum_{i=1}^{N} \sum_{j=1}^{M} E_{i}^{L}
\]

s.t. C1, C2, C3, C4, C5

Note that the last term \(\sum_{i=1}^{N} \sum_{j=1}^{M} E_{i}^{L}\) of (6) is independent of \(\alpha\). Hence, we only need to consider the problem

\[
\mathcal{P} : E_{\text{opt}} = \max_{\alpha} \sum_{i=1}^{N} \sum_{j=1}^{M} [\alpha_{ij} \times (E_{i}^{L} - E_{i}^{off})]
\]

s.t. C1, C2, C3, C4, C5

A. NEURAL NETWORK FOR THE MMKP PROBLEM

Suppose that there are two sequences, one of which is represented as a sequence of \(N\) MDs in a 3-dimensional space \(s_{1} = \{e_{i}, f_{i}^{L}, r_{i}^{f}\}_{i=1}^{N}\), where \(e_{i}, f_{i}^{L}\) and \(r_{i}^{f}\) represent the energy consumption, assigned computational resources and transmission rates, respectively. The other is represented as a sequence of \(M\) ESs in a 2-dimensional space \(s_{2} = \{F_{j}, B_{j}\}_{j=1}^{M}\), where \(F_{j}\) represents the total allocated computational resources in ES \(j\), and \(B_{j}\) represents the maximum transmission rate of ES \(j\). Our objective is to find a permutation of \(\pi = (\pi_{md}, \pi_{es})\), where \(\pi_{md}\) denotes a permutation of MDs and \(\pi_{es}\) denotes a permutation of ESs, that minimizes the sum of energy consumption values of MDs subject to the constraints of total computational resources and transmission rate. We denote the set of tasks offloaded to the set of MDs by \(C_{s_{1},s_{2}} = \{C_{1}, C_{2}, \cdots, m(s_{1}, s_{2})\}\), where \(C_{s_{1},s_{2}}\) is a sequence of \(m(s_{1}, s_{2})\) indices, each between \(1\) and \(N\). We define \(e_{\pi(i,j)} = E_{\pi(i,j)}^{L} - E_{\pi(i,j)}^{off}\), and the energy consumption given by (8) of \(N\) MDs is

\[
E_{\pi(i,j)}^{opt} = \sum_{\pi(i,j)=C_{1}}^{m(s_{1}, s_{2})} e_{\pi(i,j)}
\]
$p(\pi \mid s_1, s_2)$ is the probability of an energy consumption $E_{opt}$. Our objective is to learn a stochastic policy that, given the input set of MDs and ESs, assigns high probabilities to maximize the energy consumption $E_{opt}$ (i.e., minimize the total energy consumption of MDs). The neural network decoder uses the chain rule to formulate this objective as

$$p(\pi \mid s_1, s_2) = \prod_{i=1}^{N} p(\pi(i, j) \mid \pi(< i, j), s_1, s_2)$$  \hspace{1cm} (11)

1) ARCHITECTURE OF THE NEURAL NETWORK

Our work is inspired by the study of Pointer Networks [26], which is a modification of the sequence-to-sequence neural network. This approach modifies the attention mechanism and uses a pointer to select a member of the input sequence as the output. Pointer Networks can be applied to solve the TSP problem has not only the input sequence of MDs but also ESs. It is inappropriate for solving multidimensional multiple knapsack problem because our problem has not only the input sequence of MDs but also ESs. Accordingly, we extend the architecture of Pointer Networks as shown in Fig.2.

Distinct from the architecture of Pointer Networks, our neural network is composed of 3 RNN neural network modules, 2 encoders and 1 decoder. We set a sequence of ESs to be the input of Encoder 1, and set a sequence of MDs to be the input of Encoder 2. Encoder 1 has $M + 1$ steps, and Encoder 2 has $N + 1$. At each step, both Encoder 1 and Encoder 2 embed the input sequence into $d$ dimensions, as indicated by $\{enc_1\}_{i=0}^{M}$ and $\{enc_2\}_{i=0}^{N}$, and supply them as input to the LSTM cell. The decoder, first receives the hidden cell state of the final step of Encoder 2, as the LSTM cell input; symbol $\parallel$ indicates the start or the end of a sequence. At each step of the decoder network, one of the outputs of Encoder 2 is selected as the input for the next step, and the output is a $d$-dimensional set $\{dec\}_{j=0}^{N}$. The red arrow in Fig.2 demonstrates our pointer mechanism. For example, on the red arrow mark, the output of the decoder at step 0 points to index 2 of Encoder 1 and index 3 of Encoder 2. Hence, our neural network indicates that we should choose to offload the task on MD 3 to ES 2.

2) POINTING MECHANISM

We believe that the pointer mechanism modified by the attention mechanism is very suitable for solving combinatorial optimization problems; however, The Pointer Networks architecture can only point to one sequence element, which makes it difficult to solve more complex combinatorial optimization problems. Therefore, when solving the problem of this paper, we will perform the attention operation not only between the input of Encoder 1 (ESs) and the Decoder, but also between the input of Encoder 2 (MDs) and the Decoder.

Task (MD) pointing mechanism: The two attention matrices denoted by $W_{ref2}$ and $W_{q2} \in \mathbb{R}^{d \times d}$ and an attention vector $v \in \mathbb{R}^{d}$ are learnable parameters of the neural network. Our task attention function takes $enc_2$ (for $i = 1, 2, \cdots, N$) and $dec_j$ (for $j = 1, 2, \cdots, N$) as input to predict a distribution. The distribution represents the degree to which the model is pointing to $enc_2$ upon querying $dec_j$.

$$u_{md}^i = \begin{cases} v^T \cdot \tanh(W_{ref2} \cdot enc_2 + W_{q2} \cdot dec_j), & \text{if } i \neq \pi_{md}(j) \text{ for all } j < i. \\ -\infty, & \text{otherwise.} \end{cases}$$ \hspace{1cm} (12)

$$p(\pi_{md}(j) \mid \pi_{md} < j), s_2) = \text{softmax}(u_{md})$$ \hspace{1cm} (13)

In the scenario we study, we allow a task to be offloaded only to one ES. A task that has been offloaded cannot be transmitted to another ES anymore. Otherwise, we set the probability of its selection to $-\infty$. The computational complexity at inference time becomes $O(N^2)$ because the length of Encoder2 and Decoder is equal to $N + 1$.

ES pointing mechanism: The two attention matrices denoted by $W_{ref1}$ and $W_{q1} \in \mathbb{R}^{d \times d}$ and an attention vector

FIGURE 2. Illustration of Mptr-Net architecture.
\( v \in \mathbb{R}^d \) are learnable parameters of the neural network. Our ES attention function takes \( \text{enc}_k \) (for \( k = 1, 2, \ldots, M \)) and \( \text{dec}_j \) (for \( j = 1, 2, \ldots, N \)) as input to predict a distribution. The distribution represents the degree to which the model is pointing to \( \text{enc}_k \) upon querying \( \text{dec}_j \).

\[
u^k_k = \begin{cases} v^T \cdot \tanh (W_{\text{ref}_k} \cdot \text{enc}_k + W_{q_k} \cdot \text{dec}_j), & \text{if constraints of } C1, C2 \text{ are satisfied.} \\ \infty, & \text{otherwise.} \end{cases}
\]

(14)

\[
p(\pi_{es}(k) | \pi_{es} < k), s_1) = \text{softmax}(u_{es})
\]

(15)

Unlike a task pointer, an ES pointer can repeatedly point to an element as long as its corresponding ES has sufficient capacity (no greater than \( F_j \) and \( B_j \)). Otherwise, we set the probability to \(-\infty\).

3) POLICY-BASED REINFORCEMENT LEARNING METHOD

Our neural network is trained using a reinforcement learning method [30]. We denote by \( \theta \) the parameter of our neural network. Our training objective in solving the offloading problem entails the expected energy consumption of MDs that, given inputs \( s_1 \) and \( s_2 \), is defined as

\[ J(\theta \mid s_1, s_2) = E_{\pi \sim p_{\theta}(\cdot \mid s_1, s_2)}[E_{\text{opt}}(\pi \mid s_1, s_2) - b(s_1, s_2)\nabla_{\theta} \log p_{\theta}(\pi \mid s_1, s_2)] \]

(17)

During training, we set \( E_{\text{opt}}(\pi \mid s_1, s_2) \) of equation (10) as the reward signal, and set \( \nabla J(\theta) \) as the total training objective, where \( J(\theta) = E_{\theta_{\text{opt}}} - s_1, s_2 \cdot s_1, \text{distribution of } \text{ESS}, \text{and } s_2 \text{ is the distribution of tasks on MDs.}

We use a formulation of the reinforcement algorithm proposed by Williams [31]. This algorithm is based on the policy gradient method, and can optimize the neural network parameters by training with stochastic gradient descent. We obtain the gradient of (10) as follows:

\[
\nabla J(\theta \mid s_1, s_2) = E_{\pi \sim p_{\theta}(\cdot \mid s_1, s_2)}[(E_{\text{opt}}(\pi \mid s_1, s_2) - b(s_1, s_2))\nabla_{\theta} \log p_{\theta}(\pi \mid s_1, s_2)]
\]

(17)

By randomly generating \( G \) i.i.d. samples \( s_1^1, s_2^1, \ldots, s_1^B \sim S_1 \) and \( s_1^1, s_2^1, \ldots, s_2^B \sim S_2 \), we use Monte Carlo sampling to approximate \( \nabla J(\theta) \) as follows:

\[

\nabla J(\theta) \approx \frac{1}{B} \sum_{i=1}^{B} [(E_{\text{opt}}(\pi_i \mid s_1^i, s_2^i) - b(s_2^i))\nabla_{\theta} \log p_{\theta}(\pi_i \mid s_1^i, s_2^i)]
\]

(18)

where \( b(s_2) \) denotes the baseline value of energy consumption needed to reduce the variance of gradients. In our neural network, the baseline does not depend on \( \pi \), and we use an auxiliary network, called critic and parameterized by \( \theta_c \), to predict it as in [32]. The value of prediction \( b_0(s_2) \) depends only on input of \( s_2 \). Hence, another objective for improving the accuracy of prediction is formulated as

\[
L(\theta_c) = \frac{1}{B} \| b_0(s_2) - E_{\text{opt}}(\pi_i \mid s_1^i, s_2^i) \|_2^2
\]

(19)

where \( \| \cdot \|_2 \) denotes the \( L_2 \) norm. Our training algorithm designed for the MMKP problem is shown in Algorithm 1.

Algorithm 1 MMKP Training

Require: training set \( S_1 \), training set \( S_2 \), number of training steps \( T \), number of Encoder I steps \( T_1 \), batch size \( B \)

Ensure: Mptr-Net’s parameter \( \theta \)

1: Initialize the Mptr-Net’s parameter \( \theta \)
2: Initialize the critic network’s parameter \( \theta_c \)
3: for \( t = 0 \) to \( T \) do
4: \( s_1^i \sim \text{SAMPLEINPUT}(S_2), s_1^i \sim \text{SAMPLEINPUT}(S_1), i \in \{1, 2, \ldots, B\} \)
5: for \( j = 0 \) to \( T_1 \) do
6: \( \text{Let remainF}_j = s_1^i, F_j \) and \( \text{remainB}_j = s_1^i, B_j \) for \( i \in \{1, 2, \ldots, B\} \)
7: end for
8: \( \pi_i \sim \text{SAMPLESOLUTION}(p_0(. \mid s_2^i)) \) for \( i \in \{1, 2, \ldots, B\} \)
9: if \( \text{remainF}_j \geq \pi_i, f_j^R \) and \( \text{remainB}_j \geq \pi_i, r_j^R \) then
10: \( K_i \sim \text{SAMPLESOLUTION}(p_0(. \mid s_1^i)) \) for \( i \in \{1, 2, \ldots, B\} \)
11: \( \text{remainF}_j = \text{remainF}_j - \pi_i, f_j^R \)
12: \( \text{remainB}_j = \text{remainB}_j - \pi_i, r_j^R \)
13: \( b_i = b_0(s_2^i) \) for \( i \in \{1, 2, \ldots, B\} \)
14: \( g_0 = \frac{1}{B} \sum_{i=1}^{B} [(E_{\text{opt}}(\pi_i \mid s_1^i, s_2^i) - b_i)\nabla_{\theta} \log p_{\theta}(\pi_i \mid s_1^i, s_2^i)] \)
15: \( L_v = \frac{1}{B} \| b_0 - E_{\text{opt}}(\pi_i \mid s_1^i, s_2^i) \|_2^2 \)
16: \( \theta = \text{ADAM}(\theta, g_0) \)
17: \( \theta_c = \text{ADAM}(\theta_c, \nabla_{\theta_c} L_v) \)
18: end if
19: end for
20: return \( \theta \)

In general, the complexity of deep learning is difficult to accurately describe. The space complexity of Algorithm 1 mainly depends on the amount of parameters of Mptr-Net model. We denote \( am_{\text{listam}}, am_{\text{att}}, am_{\text{aux}} \) by the amount of parameters of each LSTM, attention matrix, and the auxiliary network, respectively. The space complexity of the model is roughly indicated as \( (2N+M) \times am_{\text{listam}} + am_{\text{att}} + am_{\text{aux}} \). \( N \) is the number of MDs, \( M \) is the number of ESs.

B. SEARCHING FOR FEASIBLE SOLUTIONS

Compared to the time spent on training, the time needed to make predictions using neural networks is generally shorter, which is beneficial for real-time applications of edge computing offload. There are four main strategies for searching results: greedy search, sampling, beam search [29], and active search [32]. Greedy search directly selects the output with the highest probability, and the respective inference time is the shortest. Sampling, based on the greedy search, extracts multiple candidate results and subsequently selects the best result as the output. Beam search selects the first few results with higher probabilities as candidate outputs at each step. Active search optimizes the results for a specific input with high accuracy. Our search algorithm chooses to combine greedy search with sampling to obtain results faster. The search
Algorithm 2 Searching for Feasible Solutions

Require: input $s_1$, input $s_2$, network parameter $\theta$, number of candidate solutions $K$, batch size $B$

Ensure: $rs$ [Feasible solution]

1: $E_i^{opt} \leftarrow E_{opt}(\pi | s_1, s_2)$
2: $n \leftarrow \lceil K/B \rceil$
3: $T_1 \leftarrow$ length of $s_1$
4: $T_2 \leftarrow$ length of $s_2$
5: for $t = 1$ to $n$ do
6: \quad $\pi_i \sim$ Samplesolution($p_{\theta}(\cdot | s_1, s_2)$ for $i \in \{1, 2, \ldots, B\}$
7: \quad $t \leftarrow \text{Argmax}(E_{opt}(\pi_1 | s_1, s_2), \ldots, E_{opt}(\pi_B | s_1, s_2))$
8: \quad for $i = 1$ to $T_1$ do
9: \quad \quad for $j = 1$ to $T_2$ do
10: \quad \quad \quad if $a_{ij} = 1$ and $(D_j/(f_i^R) + C_i/(f_i^L)) \leq T_i^{max} - C_i/(f_i^L)$ then
11: \quad \quad \quad \quad result $\leftarrow E_i^{opt}$
12: \quad \quad end if
13: \quad end for
14: end for
15: end for
16: return $rs$

The algorithm is presented in Algorithm 2. It is obviously that the computational complexity of feasible search algorithm is $O([K/B]NM)$. The increase in the time complexity of the algorithm is mainly caused by constraints check.

V. SIMULATION AND DISCUSSION

We consider the scenario with $M$ ESs in a base station such that the computation capability $F_j$ of each server $j$ is 10 gigacycles/s and channel bandwidth is $B_j = 10$ MB/s. To guarantee the minimum bandwidth and limit the maximum bandwidth, the bandwidth used for task transmission is modeled as being uniformly distributed in the range of $[0.5, 2]$ MB/s. The MD’s task transmission power is set to $P_{transfer} = 100$ mW, and the power consumed while waiting for results is $P_{wait} = 10$ mW [33]. Without a loss of generality, we assume that the data size of the computing being offloaded and the total number of CPU cycles of tasks are randomly assigned, $D_i \sim U(100, 400)$ KB and $C_i \sim U(1000, 4000)$ megacycles (10 megacycles/KB), much like [13] done. The computational capability $f_i^L$ of each MD for task $j$ is assigned from a set of $[0.5, 0.6, 0.7, 0.8]$ GHz/s. We set the least computational capability needed by task $j$ to $F_i^L = 0.5$ GHz/s, and hence set $T_i^{max} = C_i/F_i^L$. If task $i$ is offloaded to ES $j$, the latter is assigned a predetermined computational capability $f_j^R = \beta f_i^L$, $\beta \sim U[5, 10]$.

We set the input dimension of Encoder1 to 2 and the input dimension of Encoder2 to 3, and embed the input sequence of ESs and MDs into 128 dimensions by setting the number of LSTM hidden units to 128. The batch size is set to 256. The Adam optimizer is adapted to train our models. A learning rate of $10^{-3}$ is initially used during training. We apply a decay every 5000 steps by the factor of 0.96, as done in [34]. We use one glimpse [35] in the pointing mechanism to obtain more information inside the input sequences.

A. NEURAL NETWORK TRAINING

The neural network was trained on a PC made by HP and equipped with a 3.60 GHz CPU and a GPU with 4G of RAM. Our program was written in Python 3.6, and the neural network was built using TensorFlow 1.13.

We first study the impact of the problem’s computational complexity on model training. We implement a solution of the two-dimensional knapsack problem using the Pointer Networks. We set number of epochs to 10000. We observe that after 5000 steps (model training took 40 minutes) the solution of the test is very close to the optimal solution. Next, we implement a solution of the two-dimensional two-knapsack problem using our neural network. In the case of the number of epochs also set to 10000, the model trained over 10000 steps (that took 85 minutes) achieves the same effect as the former model. The curves of reward and loss are shown in Fig.3.

We observe that as the complexity of the problem increases, so does the number of epochs needed for training. If the number of epochs is insufficient or the input training solution data are not within the distribution range of the feasible solution, the maximum value of reward cannot be increased with the increase of the number of steps, hence the model training fails.

We test the impact of the number of MDs and the number of ESs on model training. We consider various input lengths and subsequently train the neural network so that each trained model can achieve an accuracy of more than 98% in the test. The training steps required for each model are shown in Table 2.
TABLE 2. Relationship of epoch (the number of training steps), the number of MDs, and the number of ESs.

| MD   | edge | epoch |
|------|------|-------|
| 5    | 15   | 25    | 30   |
| 10000| 10000| 15000 | 30000|

We find that the complexity of training is mainly determined by the number of MDs. The more MDs there are, the greater the required number of training steps and the longer the training time. The number of ESs has little effect on the number of training sessions.

B. ACCURACY OF THE NEURAL NETWORK

To evaluate the accuracy of the neural network, we first use the mixed integer programming [36] (MIP) solver of or-tools to find the optimal solution. Or-tools is an optimization algorithm package developed by Google, which can call multiple solvers through a program interface. We generate input to the Mptr-Net in the scenario of 20 MDs and 3 ESs, and use the optimal solution found by the MIP solver as the baseline. Afterwards, we compare the solutions predicted by greedy search and sampling search with the optimal solution. It is shown as Fig.4. Blue bars show the proportional relationship between results obtained using the greedy search method and the optimal solution. The horizontal axis represents the ratio, and the vertical axis is the number of solutions that are within a certain interval. We perform 100 tests. Each input is randomly generated according to the settings in the experiment. The figure shows that most of solutions are within the interval of 0.8−1, but there is still a small part of the solutions within the interval of less than 0.8. All scales in the figure are not greater than 1, which matches our goal exactly. Red bars show the proportional relationship between results obtained by using the sampling search method and the optimal solution. The approach of the sampling search method is to select a certain number of candidate solutions for the same input, and then select the maximum value of the candidate solutions as the output result. We note that the accuracy of this method is significantly higher than that of greedy search. The ratio of approximately 90% falls to the position of 1, and all proportions are within the interval of 0.9−1.

We test the relationship between the number of candidate solutions and the accuracy in the sampling search mode. To facilitate statistical analysis, we use the minimum value of the sampled solutions to compare with the optimal value. The results are shown in Table 3.

TABLE 3. Relationship of the number of candidates and prediction accuracy.

| Number | Min (ratio) | Avg (ratio) |
|--------|-------------|-------------|
| 2      | 0.642       | 0.925       |
| 4      | 0.647       | 0.956       |
| 8      | 0.793       | 0.971       |
| 16     | 0.867       | 0.987       |
| 32     | 0.815       | 0.992       |
| 64     | 0.943       | 0.994       |
| 128    | 0.954       | 0.998       |
| 256    | 0.946       | 0.998       |

The table shows that if the number of candidate solutions is small, the ratio of the worst result of the sample to the optimal solution is approximately 0.6 (the average value is still above 0.9), which is quite different from the optimal result. If the number is increased above 8, the ratio of all solutions to the optimal solution can be controlled within the interval of 0.8 to 1, and the effect is improved significantly. When the number reaches or exceeds 64, the improvement of the solution effect of the neural network tends to be slight.

C. FEASIBILITY OF SOLUTIONS

In order to evaluate the feasibility of solutions predicted by Mptr-Net, we introduce an index called missing ratio, defined as

$$\text{missing ratio} = \frac{\text{number of infeasible solutions}}{\text{number of solutions predicted}}$$

We set the value of number of solutions predicted to 100 (perform 100 tests per input), and change the solution distribution by setting the value of $\kappa$. We test under the conditions of 15 MD and 2ES, 20MDs and 3ESs, 30MDs and 6 ESs (input scale of the problem is the same as trained Mptr-Net). Result is shown in Fig.5. The missing ratio is close to 1 when the $\kappa$ value is small. As the $\kappa$ value increases, the missing ratio gradually decreases. After the $\kappa$ increases to a certain degree, the missing ratio of three neural network models in Fig. 5 are close to 0. We use the MIP solver to find the optimal solution and compared it with the solution predicted by Mptr-Net. In the region where the missing ratio is close to 1, the optimal offloading vector $\alpha$ is 0, all tasks should be performed locally. Unfortunately, the Mptr-Net cannot recognize the case of all-local execution, and cannot distinguish the lower bound of $E_{\text{opt}}$ well (in fact, outputs of Mptr-Net’s $E_{\text{opt}}$ are negative). In the area where
FIGURE 5. The missing ratio of solutions predicted by Mptr-Net in different input scales.

The missing ratio is gradually decreasing, the $\alpha$ solved by MIP shows that offloading strategy gradually changes from all-local execution to partial offloading or full offloading. We observe that the missing ratio is continuously reduced to 0 approximately. This shows that although the Mptr-Net cannot predict the lower bound of $E_{opt}$, the feasibility of solutions can be guaranteed in case of partial offloading and full offloading strategies.

D. PREDICTION TIME OF THE NEURAL NETWORK

We design a comparative experiment to assess the time needed to obtain predictions. On the one hand, the MIP solver in or-tools is utilized to find the optimal solution of the problem. On the other hand, the trained Mptr-Net is used to predict the optimal solution of the problem, and the times needed to obtain the solution are compared. We set the number of candidate solutions to 64 to ensure higher accuracy. The experiment is repeated with 20 MDs offloaded to 3 ESs and 20 MDs offloaded to 6 ESs. The results are shown in Fig.6. The $\times$ marks in the figure indicate the inference times of the neural network, and the asterisks indicate the time needed by the MIP solver to find an optimal solution. The horizontal axis indicates that the number of observations is 50. As the previous discussion shows, the problem we are studying is NP-hard. The time needed by the neural network (blue $\times$ marks) is greater than that of the MIP Solver (green asterisks) in the case of 20 MDs and 3 ESs, indicating that the computational time of the traditional solver, such as the MIP solver, is less than that needed by the neural network if the input size is small. The prediction time of Mptr-Net does not vary much in the two cases (blue and pink $\times$ marks), and is almost fixed in every instance. However, the performance of the MIP Solver begins to decline with the increase of the input scale. First, the computation time of the MIP solver is significantly longer than that of the neural network for some observations. Second, the variance of the computation time of observations increases dramatically and is sometimes difficult to control. In the 5G edge computing environment, a practical offload decision system not only needs to guarantee low latency but also requires a stable delay. The neural network can yield predictions stably after training (regardless of the complexity of the problem), which makes it very suitable for automation of decision-making jobs.

E. COMPARISON WITH NON-EXACT ALGORITHMS

1) COMPARISON WITH THE BASELINE HEURISTIC ALGORITHM

We next compare a heuristic algorithm baseline with the neural network. The detail of the heuristic algorithm baseline is shown in Appendix A. In the case of 20 MDs and 3 ESs, the number of candidate solutions to the neural network is set to 64. The input is randomly generated. A heuristic algorithm and the Mptr-Net are subsequently utilized to solve the MMKP problem with constraints. The number of tests is 100. In Fig.7, the x-axis represents the ratio of the number of solutions found by the heuristic algorithm (or the solutions predicted by Mptr-Net) to the optimal solutions found by the MIP solver, and the y-axis represents the count of ratio intervals. Orange bars represent results of the neural network, and blue bars indicate those of the heuristic algorithm. A scatter graph inside Fig.7 indicates the comparison of execution time consumed by heuristic algorithm and Mptr-Net (orange indicate Mptr-Net, blue indicate heuristic baseline).

FIGURE 6. Comparison of prediction times of the exact MIP solver and Mptr-Net for various input scales.

FIGURE 7. Solution founded by heuristic algorithm or Mptr-Net/Optimal Solution.
It is obviously that the baseline heuristic algorithm is faster than our neural network approach. However, the solution quality of heuristic baseline is very poor. We observe that the solutions predicted by the neural network are very close to the optimal solutions, and less than 5% of the solutions are near the ratio of 0.9. While, only about 70% of solutions of the heuristic algorithm are close to the optimal solutions; some solutions deviate significantly from the optimal solution, and there are solutions with ratios less than 0.4. The accuracy of the neural network is improved by over 25% compared to that of the heuristic baseline.

2) COMPARISON WITH META-HEURISTIC ALGORITHMS
It is generally believed that meta-heuristic algorithm adopts general heuristic strategy and is problem independent. We use python to implement three baseline meta-algorithms, which are Tabu Search, Simulate Annealing and Genetic Algorithm. For details, see Appendix B. We compare the Mptr-Net with three baselines in different input scales. For comparison, we use a fixed input instance in every input scale. The result is shown in Table 4.

**TABLE 4.** Mptr-Net (128 candidate solutions) vs meta-heuristic baselines in a given 5 minutes. Symbol N indicates number of MD (task), M is number of ES. The Mptr-Net spent a fixed prediction time about 59ms, other is a feasible solution searching time.

| Method                | $E_{opt}$ | Gap  | Time  |
|-----------------------|-----------|------|-------|
| N=3, M=3              | OR Tools (MIP) | 76.34 | 0.00% | 7ms    |
|                       | Mptr-Net   | 76.34 | 0.00% | 2.5s   |
|                       | Tabu Search | 76.34 | 0.00% | 3.5s   |
|                       | Simulate Annealing | 76.34 | 0.00% | 1.6s   |
|                       | Genetic Algorithm | 76.34 | 0.00% | 460ms  |
| N=10, M=3             | OR Tools (MIP) | 12.293 | 0.00% | 15ms   |
|                       | Mptr-Net   | 12.293 | 0.00% | 3.0s   |
|                       | Tabu Search | -     | -     | -      |
|                       | Simulate Annealing | 12.290 | 0.02% | 4.7s   |
|                       | Genetic Algorithm | 9.413 | 23.4% | 66s    |
| N=20, M=3             | OR Tools (MIP) | 164.917 | 0.00% | 49ms   |
|                       | Mptr-Net   | 164.917 | 0.00% | 3.1s   |
|                       | Tabu Search | -     | -     | -      |
|                       | Simulate Annealing | 92.577 | 43.9% | 16.4s  |
|                       | Genetic Algorithm | -   | -     | -      |

Computation time and gap increase as the offloading problem scale increases except Mptr-Net. In our experiment, Tabu Search baseline has the worst performance and can only converge when the input scale is very small (N = 3, M = 3). Performance of Genetic Algorithm is also poor. As the computation time increases significantly, we can obtain a sub-optimal solution with a gap of 23.4% in a medium-sized input scale. It is observed that the Genetic Algorithm does not converge in the case of large-scale input (N = 20, M = 3). The performance of Simulated Annealing algorithm is relatively better, and can still converge under large-scale input. Unfortunately, the quality of the solution is significantly reduced, with a gap of 43.9%. Compared with the heuristic algorithm approach, the meta-heuristic algorithm baseline is more general. However, the performances of the meta-heuristic algorithms are relevant to the setting of parameters. The workload of optimization is relatively heavy.

**F. GENERALIZATION OF NEURAL NETWORK**
A major criticism of deep neural networks entails the long training time; additionally, a given trained model is only applicable to one input scale. Therefore, we use a policy-based reinforcement learning algorithm to train our network. Fig.8 indicates a comparison of accuracy using the same trained model to predict different input scale problems. There is a red histogram in the middle part, where the input scale of the trained model is 20 MDs and 3 ESs, the same as the test problems’s input scale. It shows that the quality of the predicted solution is high. The probability of the ratio (the predicted solution divide the optimal solution) equal to 1 is greater than 95%. All ratios are concentrated in the interval of 0.9-1. In the left part of the Fig.8, there are two histograms, where the trained model of the middle is used.
to predict smaller input scales of 10 MDs, 2 ESs, and input scale of 10 MDs and 3 ESs, respectively. We observe that the predictive performance of Mptr-Net hardly deteriorated in these testing cases. There are two histograms in the right part, where the input scales are larger than the input scale of trained model of the middle part. We still use the trained model of the middle to predict solutions of problems in this part. Under the input scale of 30 MDs and 6 ESs, the quality of the predicted solutions is hardly reduced. With the input scale increasing to 30 MDs and 6 ESs, the quality of the solution encounters significantly decreasing, but the ratio of all predicted solutions to the optimal solution can still be controlled within the range of 0.8-1. This further illustrates that the training method of reinforcement learning helps the neural network to learn the heuristic law of the algorithm.

VI. CONCLUSION

In this paper, a neural network architecture called Mptr-Net is proposed to solve a mobile edge computation offloading problem, which is formulated as a multidimensional multiple knapsack problem with constraints. A reinforcement learning-based algorithm is designed to train the network to solve the MMKP problem. Another algorithm is proposed to search the predicted solutions for feasible solutions that satisfy the constraints. We evaluate the performance of the Mptr-Net in terms of neural network training, accuracy of predictions, time needed for inference and generalization performance. The simulation results show that the Mptr-Net can achieve a probability of over 99% of obtaining an optimal solution. The prediction time needed by Mptr-Net remain stable, and is independent of problem instance. However, neural networks have the disadvantage of having to enforce the constraints of the problem, and needing a long time for training. This will be the focus of our future research.

APPENDIXES

APPENDIX A

BASELINE HEURISTIC ALGORITHM

To the best of our knowledge, the multidimensional multiple knapsack problem has not been specifically addressed in the literature. The study of [10] considers solving an MMKP problem and proposes a heuristic algorithm. For comparison with the neural network solution, we restate the heuristic algorithm and apply it to our problem. Define the efficiency of task \( f_i \) as

\[
es_i = \frac{E_i^L - E_i^{off}}{\sum_{j=1}^{M} [f_i^R(\sum_{i=1}^{N} f_j^R - F_j)^+ + f_j^R(\sum_{i=1}^{N} f_i^R - B_j)^+]}
\]

(20)

where \([x]^+ = \max(x, 0)\). We assume that the available resources cannot meet the requirements of all tasks, i.e., \(\sum_{i=1}^{N} f_i^R > F_j\) and \(\sum_{i=1}^{N} f_i^R > B_j\). Efficiency \(e_i\) is positive and finite; otherwise, all tasks can be offloaded to ESs.

For ES \( j \), let \( r_j(t) \) denote the remaining resources of ES\( j \) at time \( t \).

\[
r_j(t) = F_j(t) \times B_j(t)
\]

(21)

where \( F_j(t) \) and \( B_j(t) \) denote the remaining capacity of computational resources and transmission rate resources, respectively, at time \( t \). The heuristic algorithm for the MMKP problem is shown as Algorithm 3.

APPENDIX B

BASELINE META-HEURISTIC ALGORITHM

This section describes the specific implementation of meta-heuristic algorithms for our MEC Offloading problem. Meta-heuristic algorithms use general heuristic strategies. However, it is still difficult to describe the problem and to deal with constraints. We solve the problem based on the combination of meta-heuristic algorithm and 0-1 integer programming. The variable of the problem is defined as \( \alpha = [\alpha_1, \alpha_2, \cdots, \alpha_{NM}]\), \( \alpha_i \in \{0, 1\} \). The optimization objective is the same as the \( E_{opt} \) in (7). In order to control the quality of the solution, we denote by \( I \) to evaluate the degree of infeasibility. It is shown as (22).

\[
I = \sum_{j=1}^{M} \max \{0, \alpha_{ij} \times (\frac{D_i}{f_j^R} + C_i) + (1 - \alpha_{ij}) \times \frac{C_i}{f_j^R} - T_i^{max}\}
\]

\[
+ \sum_{j=1}^{M} \max \{0, \sum_{i=1}^{N} \alpha_{ij} r_i^R - F_j\}
\]

\[
+ \sum_{j=1}^{M} \max \{0, \sum_{i=1}^{N} \alpha_{ij} r_i^R - B_j\}
\]

\[
+ \sum_{i=1}^{N} \max \{0, \sum_{j=1}^{M} \alpha_{ij} - 1\}
\]

(22)

where \( \alpha_{ij} \) is converted to 1-dimensional variable \( \alpha_{i+Mj} \).

If \( I = 0 \), the solution searched is feasible.

1) TABU SEARCH

Tabu Search algorithm uses tabu table to assist in finding the global optimal solution. We set \( \alpha = 0 \) as the initial solution, and get a new solution by flipping a single neighbourhood element in \( \alpha \). The value of I is calculated to verify the feasibility of a new solution. The algorithm then observes whether the new solution is in the tabu table, and calculates \( E_{opt} \) as the objective function. If the new solution is better, it updates the best solution and adds the new solution to tabu table. We set tabu length to 8, and the number of iteration to 12.

2) SIMULATE ANNEALING

The implementation of the Simulated Annealing algorithm is based on 0-1 integer programming. We set the initial solution as \( \alpha = 0 \), and then randomly flip an element in \( \alpha \) to obtain a new solution, and the I is computed to ensure the feasibility. We set the initial temperature to 100, the end temperature to 1, and the number of iterations per temperature to 300.
The temperature is decreased simply by multiple a coefficient of 0.7.

3) GENETIC ALGORITHM
We design two approaches for Genetic Algorithm to solve our MEC offloading problem. One approach is to use the penalty function method to combine the constraints with the objective function $E_{opt}$. However, the experimental results show that the solutions are very poor. It cannot obtain a feasible solution directly. Therefore, we adopted another approach, which is to check the feasibility (by computing $I$) of the solution during every operation of selection, crossover, and mutation. We set the population number as 50, the maximum number of iterations as 200, and the probability of mutation as 0.01.

REFERENCES
[1] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, “Mobile edge computing—A key technology towards 5G,” ETSI White Paper, vol. 11, no. 11, pp. 1–16, Sep. 2015.
[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” IEEE Internet Things J., vol. 3, no. 5, pp. 637–646, Oct. 2016.
[3] M. Yan, C. A. Chan, A. F. Gygax, J. Yan, L. Campbell, A. Nirmalathas, and C. Leckie, “Modeling the total energy consumption of mobile network services and applications,” Energies, vol. 12, no. 1, p. 184, Jan. 2019.
[4] P. Mach and Z. Becvar, “Mobile edge computing: A survey on architecture and computation offloading,” IEEE Commun. Surveys Tut., vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.
[5] X. Chen, L. Jiao, W. Li, and X. Fu, “Efficient multi-user computation offloading for mobile-edge cloud computing,” IEEE ACM Trans. Netw., vol. 24, no. 5, pp. 2795–2808, Oct. 2016.
[6] C. Shi, K. Habak, P. Pandurangam, M. Ammar, M. Naik, and E. Zegura, “COSMOS: Computation offloading as a service for mobile devices,” in Proc. 15th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.,Philadelphia, PA, USA, 2014, pp. 287–296.
[7] P. Dütting, Z. Feng, H. Narasimhan, D. C. Parkes, and S. S. Ravindranathan, “Optimal auctions through deep learning,” 2017, arXiv:1706.03459. [Online]. Available: http://arxiv.org/abs/1706.03459
[8] G. Villarrubia, J. F. De Paz, P. Chamoso, and F. D. la Prieta, “Artificial neural networks used in optimization problems,” Neurocomputing, vol. 272, pp. 10–16, Jan. 2018.
[9] J. Li, H. Gao, T. Lv, and Y. Lu, “Deep reinforcement learning based based computation offloading and resource allocation for MEC,” in Proc. IEEE Wireless Commun. Netw. Conf. (WCNC), Apr. 2018, pp. 6–11.
[10] Y. Song, C. Zhang, and Y. Fang, “Multiple dimensional knapsack problem and its applications in cognitive radio networks,” in Proc. IEEE Military Commun. Conf., Washington, DC, USA, Nov. 2008, pp. 1–7.
[11] F. Wang, J. Xu, and Z. Ding, “Multi-Antenna NOMA for Computation Offloading in Multiuser Mobile Edge Computing Systems,” IEEE Trans. Commun., vol. 67, no. 3, pp. 2450–2463, Mar. 2019.
[12] X. Yang, X. Yu, H. Huang, and H. Zhu, “Energy efficiency based joint computation offloading and resource allocation in multi-access MEC systems,” IEEE Access, vol. 7, pp. 117054–117062, 2019.
[13] F. Guo, H. Zhang, H. Ji, X. Li, and Y. C. M. Leung, “Energy efficient computation offloading for multi-access mec enabled small cell networks,” in Proc. IEEE Int. Conf. Commun. Workshops (ICC), Kansas City, MO, USA, May 2018, pp. 1–6.
[14] S. Sardellitti, G. Scutari, and S. Barbarossa, “Joint optimization of radio and computational resources for multicell mobile-edge computing,” IEEE Trans. Signal Inf. Process. Netw., vol. 1, no. 2, pp. 89–103, Jun. 2015.
[15] L. Yang, H. Zhang, M. Li, J. Guo, and H. H., “Mobile edge computing empowered energy efficient task offloading in 5G,” IEEE Trans. Veh. Technol., vol. 67, no. 7, pp. 6398–6409, Jul. 2018.
[16] B. Cao, L. Zhang, Y. Li, D. Feng, and W. Cao, “Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework,” IEEE Commun. Mag., vol. 57, no. 3, pp. 56–62, Mar. 2019.
[17] C. Zhang and Z. Zheng, “Task migration for mobile edge computing using deep reinforcement learning,” Future Gener. Comput. Syst., vol. 96, pp. 111–118, Jul. 2019.
[18] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, “Learning-based computation offloading for IoT devices with energy harvesting,” IEEE Trans. Veh. Technol., vol. 68, no. 2, pp. 1930–1941, Feb. 2019.
[19] Y. Wei, F. R. Yu, M. Song, and Z. Han, “Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor–critic deep reinforcement learning,” IEEE Internet Things J., vol. 6, no. 2, pp. 2061–2073, Apr. 2019.
[20] C. Zhang, Z. Liu, B. Gu, K. Yamori, and Y. Tanaka, “A deep reinforcement learning based approach for cost- and energy-aware multi-flow mobile data offloading,” IEICE Trans. Commun., vol. E101.B, no. 7, pp. 1625–1634, Jul. 2018.
[21] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, “Deep reinforcement learning based joint task offloading and bandwidth allocation for multi-user mobile edge computing,” Digit. Commun. Netw., vol. 5, no. 1, pp. 10–17, Feb. 2019.
[22] L. Huang, X. Feng, L. Zhang, L. Qian, and Y. Wu, “Multi-server multi-user multi-task computation offloading for mobile edge computing networks,” Sensors, vol. 19, no. 6, p. 1446, Mar. 2019.
[23] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, “Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning,” IEEE Commun. Mag., vol. 57, no. 5, pp. 64–69, May 2019.
[24] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, “Energy-optimal mobile cloud computing under stochastic wireless channel,” IEEE Trans. Wireless Commun., vol. 12, no. 9, pp. 4569–4581, Sep. 2013.
[25] D. Pisinger and P. Toth, “Knapsack problems,” in Handbook of Combinatorial Optimization. Boston, MA, USA: Springer, 1998, pp. 299–428.
[26] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer networks,” in Proc. Adv. Neural Inf. Process. Syst. Montreal, QC, Canada, 2015, pp. 2692–2700.
[27] P. Zhao, H. Tian, C. Qin, and G. Nie, “Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing,” IEEE Access, vol. 5, pp. 11255–11268, 2017.
[28] J. Zhang, W. Xia, F. Yan, and L. Shen, “Joint computation offloading and resource allocation optimization in heterogeneous networks with mobile edge computing,” IEEE Access, vol. 6, pp. 19324–19337, 2018.
[29] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in Proc. Adv. Neural Inf. Process. Syst. Montreal, QC, Canada, 2014, pp. 3104–3112.
[30] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 2018.
[31] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Mach. Learn., vol. 8, nos. 3–4, pp. 229–256, May 1992.
[32] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, “Neural combinatorial optimization with reinforcement learning,” 2016, arXiv:1611.09940. [Online]. Available: http://arxiv.org/pdf/1611.09940.pdf
[33] X. Chen, “Decentralized computation offloading game for mobile cloud computing,” IEEE Trans. Parallel Distrib. Syst., vol. 26, no. 4, pp. 974–983, Apr. 2015.
[34] H. Hu, X. Zhang, X. Yan, L. Wang, and Y. Xu, “Solving a new 3D bin packing problem with deep reinforcement learning method,” 2017, arXiv:1708.05930. [Online]. Available: http://arxiv.org/abs/1708.05930
[35] O. Vinyals, S. Bengio, and M. Kudlur, “Order matters: Sequence to sequence for sets,” 2015, arXiv:1511.06391. [Online]. Available: http://arxiv.org/abs/1511.06391
[36] H. M. Salkin, Foundations of Integer Programming. Amsterdam, The Netherlands: North Holland, 1989.
YUAN ZHANG received the B.Eng. degree in telecommunications from the Beijing University of Posts and Telecommunications, in 2008, the M.Eng. degree in electronic engineering from Tsinghua University, Beijing, China, in 2011, and the Ph.D. degree in computer science from the University of Goettingen, Germany, in 2015. She is currently an Assistant Professor with the Communication University of China. Her research interests include mobile edge computing, immersive VR, and intelligent multimedia communications.

JINYAO YAN received the B.S. degree from Tianjin University, the M.S. and Ph.D. (in engineering) degrees from the Beijing Broadcasting Institute, and the Ph.D. (in science) degree from the Swiss Federal Institute of Technology (ETH Zurich). Since 2010, he has been a Professor with the Communication University of China, Beijing, China. His research interests are in the areas of data analytics, multimedia computing and communication.