Graph Mapping Offloading Model Based On Deep Reinforcement Learning With Dependent Task

Ning Mao†, Yuanfang Chen∗, Mohsen Guizani‡, Gyu Myoung Lee§
†School of Software, Dalian University of Technology, DaLian, China
∗School of Cyberspace, Hangzhou Dianzi University, Hangzhou, China
‡Qatar University, Qatar
§Liverpool John Moores University, UK

Abstract—In order to solve the problem of task offloading with dependent subtasks in mobile edge computing (MEC), we propose a graph mapping offloading model based on deep reinforcement learning (DRL). We model the user's computing task as directed acyclic graph (DAG), called DAG task. Then the DAG task is converted into a topological sequence composed of task vectors according to the custom priority. And the model we proposed can map the topological sequence to offloading decisions. The offloading problem is formulated as a Markov decision process (MDP) to minimize the tradeoff between latency and energy consumption. The evaluation results demonstrate that our DRL-based graph mapping offloading model has better decision-making ability, which proves the availability and effectiveness of the model.

Index Terms—mobile edge computing (MEC), task offloading, directed acyclic graph (DAG), Markov decision process (MDP), deep reinforcement learning (DRL)

I. Introduction

With the increase of mobile equipments, the diversity and complexity of mobile applications are getting higher and higher, such as VR, AR, face recognition [1], [2]. Although the performance of mobile equipments has been greatly improved, many computing intensive applications can not be completed in effective time by themselves. MEC sinks the computing resources to the edge of the network closer to the user, and users use the MEC server to help complete the task computing [3]. Compared with completely local computation, it can greatly shorten the latency and reduce the energy consumption. How to offload task reasonably is one of the main problems in MEC. The offloading decision problem in MEC environment can be converted as an optimization problem, which needs to be solved in a given environment. However, due to the complexity of MEC environment, the final optimization problem is often a NP hard problem, which makes the traditional solution method no longer feasible.

Deep reinforcement learning (DRL) has been greatly developed due to its strong perception and decision-making ability [4]. It is an important research direction to introduce DRL into MEC environment to solve the problem of offloading decision and resource allocation. For example, [5] proposed a reinf orcement learning framework based on DQN to solve the resource allocation problem in edge computing and designed a real-time adaptive computing resource allocation strategy for user task to improve the average end-to-end reliability. In [6], authors consider the problem of wireless driven MEC computational offloading in a time-varying channel environment, that is, the difficult combinatorial optimization problem needs to be solved in the channel coherent time and proposed an adaptive DRL framework to complete the offloading decision approximately in real time. A vehicle edge computing network architecture is explored in [7], in which the vehicle can be used as a mobile edge server to provide computing services for the nearby users and proposed Q-learning and DQN based algorithm to maximize the long-term utility of the vehicle edge computing network. In [8], a MEC system composed of multiple mobile users with random task arrival was considered, and the target of offloading strategy is to minimize the long-term average computing cost in terms of buffer delay, and deep deterministic policy gradient (DDPG) algorithm was used to independently learn the effective task offloading strategy of each mobile user.

However, the current computing intensive task is generally composed of multiple subtasks, and there is a dependency relationship between the subtasks, so the computing task can be abstracted into DAG task. Therefore, the original task offloading decision is transformed into the offloading decisions of all task nodes in DAG task. But with the increase of the number of task nodes, it will be very difficult to get the optimal offloading decision of all task nodes [9]. Most of the existing researches are based on heuristic algorithm. For example, in [10], a heuristic algorithm for wireless sensing joint scheduling and computational offloading for multi-component applications was proposed to minimize the computation latency of the application. In [11], the practical application was modeled as a general DAG task structure, then the offloading problem was modeled as 0-1 programming problem, and the particle swarm optimization algorithm was used to solve the problem. In [12], authors considered the scenario

Corresponding author: Yuanfang Chen, yuanfang.chen.tina@gmail.com. This work was supported by the National Natural Science Foundation of China (Grant No. 61802097), the Project of Qianjiang Talent (Grant No. QJD1802020), and the Key Research & Development Plan of Zhejiang Province (Grant No. 2019C01012).

of multiple servers. The final problem was modeled as a nonlinear integer programming problem and solved by genetic algorithm.

However, using heuristic algorithm to solve the offloading decision problem of DAG task often leads to great computation cost, which causes MEC system not having real-time decision-making ability [13]. Moreover, heuristic algorithms need more or less expert knowledge, which reduces the flexibility of model design. Therefore, reinforcement learning has been widely used to solve the offloading decision problem of DAG task. [14] proposed a temporal-difference learning based algorithm to solve the problem of online task scheduling optimization and scheduling procedure was modeled as Markov decision process. In [15], authors proposed a deep reinforcement learning based multi-job dependent task offloading algorithm and the graph convolutional network(GCN) was used to extract the deep information of DAG task. [15] alse considered DAG task, the target was to jointly determine the offloading decision and resource allocation of each subtask under the time-varying wireless fading channel and random edge computing ability, and a deep reinforcement learning framework based on actor-critic learning structure was proposed to solve the problem.

In this paper, we design a DRL-based graph mapping offloading model, which makes full use of the strong perception and decision-making ability of deep reinforcement learning to sove the offloading decision problem of DAG task. Our contributions are as follows:

- In order to transform DAG task into the data that can be input into the model. We propose a priority-topology-relative-entropy (PTRE) algorithm to convert the task nodes in DAG task into a sequence of task vectors according to their priority without violating the dependency of DAG task. That is to say, DAG task can be transformed into a topological sequence by PTRE algorithm.
- We formulate the offloading decision problem of DAG task into Markov decision process, define state, action and reward, and design a DRL-based graph mapping offloading model which maps task vector into offloading decision. This model can get the approximate optimal offloading decision without any expert knowledge.
- The special actor network and critic network are designed to handle the topological sequence. The A3C deep reinforcement learning algorithm is adopted to train the model. The evaluation results show that the model is superior in performance.

The rest of the paper is organised as follows. In Section II, we present the system model in detail. In Section III, we establishe a mathematical model for practical problems. In Section IV, the details of DRL-based graph mapping offloading model will be discussed in detail. Section V is the experimental part, we present the evaluation results here. Section VI is the last section of this paper, which summarizes the full text.

II. System Model

The computing resources of MEC server are allocated in the way of virtual machine (VM) [16]. The interaction between the user equipment and the server is essentially the interaction with the corresponding VM, which provides private computing, communication and storage resources. In user equipment, the local CPU is used for local computing, the local transmission unit (LTM) is used for uploading the tasks and receiving the results of edge computing. the virtual central processor (VCPU) in VM is used for edge computing, and the virtual transmission unit (VTM) is used for receiving the uploaded tasks by user equipment and returning the results of edge computing.

A. DAG task

There is a graph parser (GP) in user equipment, which is used to convert computing intensive task into DAG task format. The target is to make offloading decisions for this kind of DAG task, as shown in Figure 1.

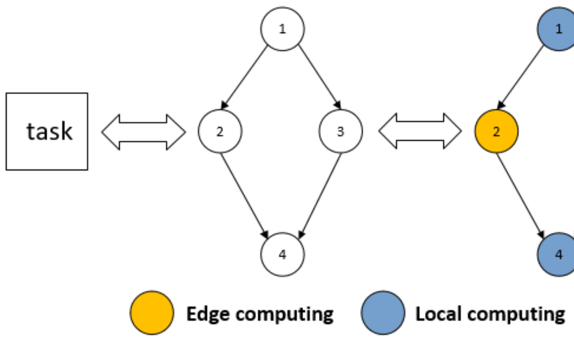

Fig. 1: DAG task format of computing intensive task.

DAG task is represented by point set N and edge set E, that is, DAG task $G = (N, E)$, task node $n_i \in N$, where $i \in 1, 2, \ldots\ldots, |N|$, $|N|$ is the total number of task nodes. Edge $e_{ij} \in E$ denotes task node n_i is the precursor of task node n_j and task node n_j is the successor of task node n_i. A task node without a precursor is called an entry task node, while a task node without a successor is called an exit task node. Each edge has two weights, the local weight w_{ij}^l and the offloading weight w_{ij}^o are used to represent the fixed communication cost of the same equipment and different equipments between task node n_i and task node n_j. Task node n_i should include the following information: the data size, d_i^{exe}, the required CPU cycles, c_i, the data size of the result after computing, d_i^{res}.

B. System Flow

Figure 2 describes the execution flow of the whole system. The user's computing task is first transformed into DAG task by GP. Then the offloading decisions of all task nodes in DAG task will be made in mapping offloading module (MOM).

- Local Computing:
  If task node n_i is decided to compute locally, then n_i is computed by the local CPU. So the computing latency and energy consumption of task node n_i can be expressed as follows:

$$\begin{cases} t_i^l = \dfrac{c_i}{f^l} \\ e_i^l = p^l t_i^l \end{cases} \tag{1}$$

  where f^l is the computing rate of the local equipment, $p^l = \sigma(f^l)^\tau$ is the local computing power, σ and τ are constants.
- Edge Computing:
  If task node n_i is decided to compute in edge server, it will go through three processes:
  First, user equipment uploads the task node n_i to the MEC server through LTM, and the corresponding VM in the server receives the task through the VTM. After that, the edge computing is carried out by VCPU. Finally, the result is sent back to the user equipment through VTM, and the user equipment receives the result through LTM.

$$\begin{cases} t_i^u = \dfrac{d_i^{exe}}{v^u}, t_i^{vm} = \dfrac{c_i}{f^{vm}}, t_i^d = \dfrac{d_i^{res}}{v^d} \\ e_i^o = p^u t_i^u + p^d t_i^d \end{cases} \tag{2}$$

  where v^u is uploading rate of user equipment, f^{vm} is the computing rate of the VM, v^d is the returning rate, p^u and p^d are the uploading and returning power of user equipment respectively.
  Whether it's local computing or edge computing, all computing components and transmission components are not allowed to preempt resources. When resources conflict, they need to wait in queue. But, the uploading process of LTM and VTM does not affect their receiving process, and vice versa.

III. Problem Formulation

In order to minimize the trade-off between total latency and energy consumption, we need to transform the actual offloading problem into a mathematical problem. The completion time of local computing, uploading, edge computing and result returning of task node n_i are respectively expressed as: ECT_i^l, ECT_i^u, ECT_i^o, ECT_i^d. If task node n_i is decided to compute locally, then ECT_i^u, ECT_i^o and ECT_i^d are meaningless and set to 0. If it is decided to offload, then ECT_i^l is meaningless and set to 0. When processing task node n_i, the available time of CPU, LTM, VCPU and VTM are expressed as: CAT_i^l, CAT_i^u, CAT_i^o, CAT_i^d. Using P(i) to represent the set of predecessor nodes of task node n_i, we can get:

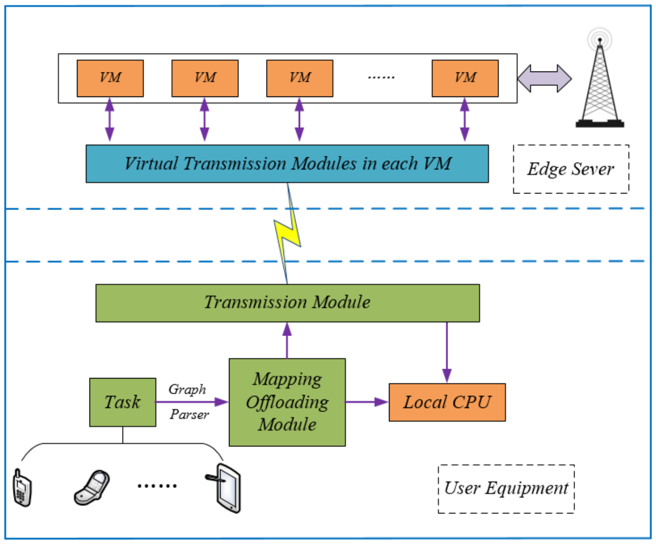

Fig. 2: The whole system flow.

$$\begin{cases} ECT_i^l = \max\left(CAT_i^l, \max\limits_{j\in P(i)}\left(ECT_j^l, ECT_j^d\right)\right) + t_i^l \\ ECT_i^u = \max\left(CAT_i^u, \max\limits_{j\in P(i)}\left(ECT_j^l, ECT_j^u\right)\right) + t_i^u \\ ECT_i^o = \max\left(CAT_i^o, \max\left(ECT_i^u, \max\limits_{j\in P(i)} ECT_j^o\right)\right) + t_i^{vm} \\ ECT_i^d = \max\left(CAT_i^d, ECT_i^o\right) + t_i^d \end{cases} \tag{3}$$

The meanings of formula (3) are :

- The premise for task node n_i to start local computing is that the CPU is available and all its precursor nodes have completed the computing. Local computing start time plus local computing latency leads to ECT_i^l.
- The premise for task node n_i to start uploading is that LTM is available and all its precursor nodes have completed computing or uploaded locally. Uploading start time plus uploading latency leads to ECT_i^u.
- The premise for task node n_i to start edge computing is that VCPU is available, uploading is completed, and all its precursor nodes have completed computing. The edge computing start time plus the edge computing latency leads to ECT_i^o.
- The premise for the result of task node n_i to start returning is that VTM is available and edge computing is completed. The start time of returning plus the returning latency leads to ECT_i^d.

The start time of DAG task is set to 0, then the last exit task node's completation time is the scheduling latency of the whole DAG task. The total energy consumption is the sum of the energy consumption of all task nodes. Let a_i represents the offloading decision of task node n_i, $a_i = 0$ indicates that the task is decided to compute locally, and $a_i = 1$ indicates that the task is decided to compute in edge server, that is:

$$\begin{cases} T_{all} = \max\left(\max\limits_{n_i \in exit}\left(ECT_i^l, ECT_i^d\right)\right) \\ E_{all} = \sum\limits_{n_i \in N, a_i = 0} e_i^l + \sum\limits_{n_i \in N, a_i = 1} e_i^o \end{cases} \tag{4}$$

Compared with all task nodes compute locally, the latency benefit G_T and energy consumption benefit G_E brought by a certain scheduling scheme are defined as follows:

$$G_T = \frac{T_l - T_{all}}{T_l}, G_E = \frac{E_l - E_{all}}{E_l} \tag{5}$$

where T_l and E_l represent the latency and energy consumption when all task nodes are local computing respectively. The smaller the T_{all} and E_{all} brought by a certain scheduling scheme, the higher the latency benefit and energy consumption benefit. Therefore, the objective function is defined as the comprehensive profit, and the ultimate target is to maximize the comprehensive, that is:

$$\max \quad CP = \alpha G_T + (1 - \alpha) G_E \tag{6}$$

where α is equilibrium factor, which takes value in the interval $(0, 1)$.

IV. DRL-based Graph Mapping Offloading Model

In this section, we describe each part of DRL-based graph mapping offloading model (DRL-GMOM). Firstly, the priority-topology-relative-entropy algorithm (PTRE) is introduced to transform DAG task into task vector sequence. The algorithm plays the role of data preprocessing and can provide training data for DRL-GMOM. The offloading decision problem is formulated as Markov decision process, and the corresponding state space, action space and reward are presented. Finally, we give the structure of DRL-GMOM in detail, and A3C algorithm is used to complete the model training.

A. Priority-Topology-Relative-Entropy algorithm

Because of the dependency between task nodes, the offloading decision order of task nodes must be a topological sort of DAG task. In this way, we can ensure that the precursor node must make decisions before the successor nodes. In this paper, the priority is calculated according to the principle that the higher the execution cost, the higher the priority. The execution cost is represented by cost quota, which is calculated by latency, energy consumption and fixed task communication cost.

- If task node n_i is decided to compute locally, then time cost, $t_{c_i^l} = t_i^l$, energy consumption cost, $e_{c_i^l} = e_i^l$, and $S(i)$ is used to represent the set of all successor nodes of task node n_i, and the local execution cost quota is:

$$q_{c_i^l} = \alpha t_{c_i^l} + (1 - \alpha) e_{c_i^l} + \beta \sum_{j \in S(i)} w_{ij}^l \tag{7}$$

- If task n_i is decided to compute in edge server, then time cost, $t_{c_i^o} = t_i^u + t_i^{vm} + t_i^d$, energy consumption cost, $e_{c_i^o} = e_i^o$, and the edge execution cost quota is:

$$q_{c_i^o} = \alpha t_{c_i^o} + (1 - \alpha) e_{c_i^o} + \beta \sum_{j \in S(i)} w_{ij}^o \tag{8}$$

From (1) and (2), we get the final cost quota: $q_i = \min(q_{c_i^l}, q_{c_i^o})$, and define the final priority as follows:

$$rank[i] = \begin{cases} \max\limits_{j \in S(i)} (rank(j)) + q_i & n_i \in exit \\ q_i & n_i \notin exit \end{cases} \tag{9}$$

Same as α, β is also equilibrium factor, which takes value in the interval $(0, 1)$. And exit represents the set of exit task nodes. According to the priority, all task nodes in DAG task are sorted in descending order, and a task sequence idx is obtained, that is, $idx[1]$ is the task node with the highest priority, and $idx[|N|]$ is the task node with the lowest priority. From formula (5), it can be concluded that the task sequence represented by idx must be a topological sequence of the original DAG task. For convenience, the symbol n_i is used later in this paper to refer to task node $idx[i]$, that is, n_i is represented as the i-th task node in the task sequence idx. Similarly, $P(i)$ and $S(i)$ are represented as the set of predecessor nodes and the set of successor nodes of the i-th task node in task sequence idx, respectively.

According to the order of task sequence idx, each task node is represented from the perspective of information. We define the task vector of task node n_i ($idx[i]$) is composed of the following information:

- Its priority index i.
- Latency of local computing t_i^l and energy consumption e_i^l.
- Each latency of edge computing t_i^u, t_i^{vm}, t_i^d and energy consumption e_i^o.
- The index of m predecessor nodes and m successor nodes.

The selected precursor and successor task nodes should contain as much information as possible that is not included in task node n_i, that is, those precursor and successor task nodes that are not similar to the task node n_i should be selected, so as to ensure that the transformed task vector contains more information. In this paper, we use relative entropy [17] to describe the dissimilarity of task nodes. The larger the relative entropy, the more dissimilar the two task nodes are. The information vector of task node n_i is defined as: $vec_i = (i, t_i^u\ , t_i^{vm}\ , t_i^d, t_i^l, e_i^o, e_i^l)$, so the relative entropy between task node n_i and n_j is:

$$KL(vec_i\,|vec_j) = -\sum_{loc=0}^{len-1} vec_i\,(loc) \log \frac{vec_j\,(loc)}{vec_i\,(loc)} \tag{10}$$

where len is the length of the information vector and loc is the element index of the information vector. The relative entropy values of the task node and all its predecessor nodes and successor nodes are calculated respectively, and then the index of the first m task nodes is extracted by descending sorting. In this way, the complete task vector $task_i$ of task node n_i is obtained, and the sequence of task vectors of all task nodes in task sequence idx is represented by $TASK$. The pseudo code of priority-topology-relative-entropy algorithm is described as Algorithm 1.

Algorithm 1 PTRE algorithm

```
Input: $t_i^l, t_i^u, t_i^{vm}, t_i^d, e_i^l, e_i^o$ of each task node $n_i$
Output: sequence of task vector: $TASK$
1: for each task node $n_i$ in DAG task do
2:     $t_{c_i^l} = t_i^l,\ e_{c_i^l} = e_i^l, t_{c_i^o} = t_i^u + t_i^{vm} + t_i^d, e_{c_i^o} = e_i^o$
3:     calculate:
       $$\begin{cases} q_{c_i^l} = \alpha t_{c_i^l} + (1-\alpha) e_{c_i^l} + \beta \sum\limits_{j \in S(i)} w_{ij}^l \\ q_{c_i^o} = \alpha t_{c_i^o} + (1-\alpha) e_{c_i^o} + \beta \sum\limits_{j \in S(i)} w_{ij}^o \end{cases}$$
4:     $q_i = \min(q_{c_i^l}, q_{c_i^o})$
5:     calculate:
       $$rank[i] = \begin{cases} \max\limits_{j \in S(i)} (rank(j)) + q_i & n_i \in exit \\ q_i & n_i \notin exit \end{cases}$$
6: end for
7: sort all task nodes in descending order by the array rank and get task sequence $idx$
8: define $idx[i]s$ information vector as:
       $$vec_i = (i, t_i^u, t_i^{vm}, t_i^d, t_i^l, e_i^o, e_i^l)$$
9: for $n_i$ in idx do
10:    for $n_j inP(i), n_k inS(i)$ do
11:        calculate:
           $$\begin{cases} KL_P[j] = \sum\limits_{n=1}^{N} vec_i(loc) \, log \frac{vec_j(loc)}{vec_i(loc)} \\ KL_S[k] = \sum\limits_{n=1}^{N} vec_i(loc) \, log \frac{vec_k(loc)}{vec_i(loc)} \end{cases}$$
12:    end for
13:    sort $KL_P$ and $KL_S$ in descending order, get $pred = KL_P[1:m]$ and $succ = KL_S[1:m]$
14:    put $task_i = (i, t_i^u, t_i^{vm}, t_i^d, t_i^l, e_i^o, e_i^l, pred, succ)$ into $TASK$
15: end for
```

B. Markov decision process(MDP)

Because each user is connected to one VM to provide private computing, communication and storage resources, the parameters of network environment are considered as static. Therefore, we will focus on the perspective of DAG task to establish Markov decision process.

- State space: State space is defined as the sequence composed of TASK and offloading decisions of historical task nodes. The decision set of historical task nodes is represented by H, then:

$$state = \{(TASK, sequence\ of\ a_i) |\ a_i \in H\} \quad (11)$$

- Action space: Because each task node can only be decided as local computing or edge computing, the action is defined as:

$$action = \{0, 1\} \quad (12)$$

where 0 represents local computing and 1 represents edge computing.

- Reward: When task node n_i is computed according to decision a_i, the increment of total latency and energy consumption is expressed as ΔT_i and ΔE_i.So the reward function can be defined as:

$$reward_i = \alpha \frac{\frac{T_l}{|N|} - \Delta T_i}{T_l} + (1-\alpha) \frac{\frac{E_l}{|N|} - \Delta E_i}{E_l} \quad (13)$$

So we can get the cumulative reward in the process of MDP: $R = \sum_{i=1}^{|N|} \gamma^i reward_i$. It can be concluded that while maximizing the cumulative reward R, the objective function CP is also maximized.

C. Model Design

The DRL-GMOM is based on sequence to sequence model [18] with attention mechanism, which is a mapping model for mapping task vector sequence into off loading decision sequence. The main structure of the actor network is composed of an encoder and a decoder, both of which use recurrent neural network. According to $TASK$, $task_i$ is input into the encoder for encoding. After receiving all the task vectors, the decoder is initialized by the encoder's final hidden state, and then begins decoding. At the final output, decision $\pi_\theta(a_i|state_i)$ is obtained through softmax layer, where θ is the parameter of the actor network.

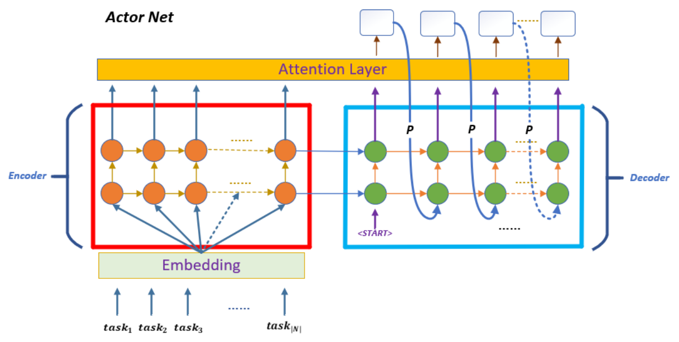

Fig. 3: Actor network in DRL-GMOM.

Critic network is used to evaluate state value $V_\omega(state_i)$, where ω is the parameter of the critic network. The main structure of the network is composed of recurrent neural network and fully connected network. Initial state is also initialized by the encoder's final hidden state in the actor network. And the historical decision is taken as the input.

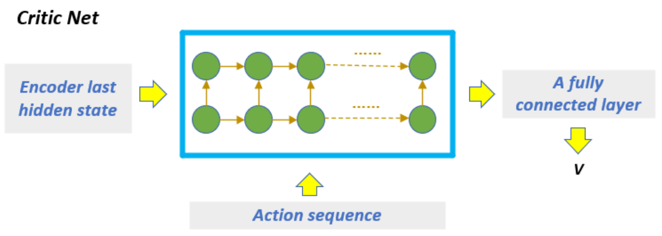

Fig. 4: Critic network in DRL-GMOM.

The final hidden state of the network is mapped to the state value through the fully connected network.

Considering that A3C algorithm interacts with the environment in multiple threads and executes multiple agents asynchronously [19], it can remove the correlation between state transition samples in the training process. A3C algorithm replaces the experience pool which needs a lot of memory for data storage by asynchronous execution. In view of the above advantages, we adopts A3C algorithm to train DRL-GMOM. The random initialization parameters of actor network and critic network in the main thread are θ_m and ω_m, and the parameters of actor network and critic network in i-th sub-thread are expressed by θ_i and ω_i respectively. The global maximum number of iterations is set to E_{max}. The training steps are described as Algorithm 2.

Algorithm 2 Training steps of DRL-GMOM in each thread

```
Input: DAG task
Output: specific offloading decision of every task node
 1: initialize Actor and Critic in main thread with random weight $\theta_m$ and $\omega_m$
 2: the weights of Actor and Critic in specific thread $i$ are $\theta_i$ and $\omega_i$
 3: initialize $E = 0,\ t = 0$
 4: repeat
 5:     $d\theta_m = 0, d\omega_m = 0, \theta_i = \theta_m, \omega_i = \omega_m, flag = t$
 6:     while $state_t$ is not terminal, $t - flag \neq t_{local}$ do
 7:         choose $a_t$ according to $\pi(a_t|state_t, \theta_i)$
 8:         execute $a_t$ and get $reward_i$ and $state_{t+1}$
 9:         $t = t + 1, E = E + 1$
10:     end while
11:     Initialize the RNN's Initial state in critic network with the last hidden state in encoder
12:     if $state_t$ is not terminal state then
13:         $R_t = V\,(state_t|\omega_i)$
14:     else
15:         $R_t = 0, state_t = state_t$
16:     end if
17:     for $j = t-1, t-2, \ldots\ldots, flag$ do
18:         $R_j = reward_j + \gamma R_{j+1},\ A_j^i = R_j - V(s_j|\omega_i)$
19:         $d\theta_m = d\theta_m + \nabla_{\theta_m} log\pi(a_j|s_j, \theta_i)A_j^i$
20:         $d\omega_m = d\omega_m + \frac{\partial(A_j^i)^2}{\partial\omega_i}$
21:     end for
22:     Update $\theta_m$ and $\omega_m$ by $d\theta_m$ and $d\omega_m$
23: until $E > E_{max}$
```

V. Performance Evaluation

A. Simulation Environment

According to the DAG graph generator provided by reference [20], the data sets required are generated. The data sets are divided into training data sets and test data sets.

Dag tasks in training sets and test sets are divided into 8 categories according to the number of task nodes: 12, 16, 20, 24, 28, 32, 36 and 40. The width and density of DAG task are generated randomly. d_i^{exe} and d_i^{res} of each task node are randomly generated in the interval [25KB, 850KB]. And the fixed communication costw_{ij}^l and w_{ij}^o are randomly generated in the interval $[1, 10]$.

There are 500 DAG tasks in each category in the training data sets and 100 DAG tasks in each category in the test data sets. In other words, there are 4000 DAG tasks in training data sets and 800 DAG tasks in test data sets. The target is to simulate computing intensive task with different number of task nodes.

The encoder and decoder in actor network and the RNN in critic network are composed of two layers of GRU, and the number of hidden layer, num_{hidden} is 512. The discount factor $\gamma = 0.99$, the global maximum iteration number $E_{max} = 4000$. The training learning rate of actor network, lr_a and critic network, lr_c both are 0.0005, and Adam is used as the optimizer for training. The uploading bandwidth and downloading bandwidth between user and edge server are 10Mbps. VM's computing rate $f^{vm} = 8 \times 10^6$ B/s, user equipment's computing rate $f^l = 1 \times 10^6$ B/s, $p^u = 1.3w$, $p^d = 1.2w$. Equilibrium factor $\alpha = 0.5$,$\beta = 0.1$. We create 4 sub-threads to interact with the environment, and set t_{local} in each sub-thread to 100.

TABLE I: Main Parameter

Parameter	Value
d_i^{exe}, d_i^{res}	[25KB, 850KB]
w_{ij}^l, w_{ij}^o	[1, 10]
num_{hidden}	512
γ	0.99
E_{max}	4000
lr_a, lr_c	0.0005
f^{vm}	8×10^6B/s
f^l	1×10^6B/s
p^u, p^d	$1.3w, 1.2w$
$\alpha,\ \beta$	0.5,0.1

In order to highlight the performance of DRL-GMOM, the following algorithms will be compared:

- HEFT algorithm: [21] HEFT algorithm is a heuristic static DAG scheduling algorithm. The core of the algorithm is based on the earliest completion time.
- Round Robin (RR): [22] Each time, the task node is assigned to the user equipment and the corresponding VM in turn, and then the cycle is restarted.

- Temporal-Difference learning (TD): [14] Compared with Monte Carlo algorithm, this algorithm can solve reinforcement learning problems without using complete state sequence. The estimation is made by TD error in each time slot.
- Random policy: Whether the task node n_i computes locally or in edge server is completely random without violating the dependency relationship.
- Greedy policy: If the comprehensive profit computed locally is greater than that computed in the server, it's decided to be local computing, otherwise it is determined to be computed in server.

B. Simulation Results

The DRL-GMOM is trained under the above experimental conditions. After the training, the performance of the 5 algorithms are compared through the test data sets. The results are shown in Table 1. It can be concluded that our DRL-GMOM has the best performance on all test data sets. And the second is Temporal-Difference learning algorithm. We get the optimal average comprehensive profit (shown in brackets) of test sets with 12, 16 and 20 task nodes. It can be found that the average comprehensive profit obtained by DRL-GMOM is very close to the optimal solution.

TABLE II: Comparison of Average Comprehensive Profit

Nodes	DRL-GMOM	HEFT	RR	TD	Random	Greedy
12	0.543(0.546)	0.280	0.320	0.428	0.314	0.292
16	0.553(0.556)	0.322	0.337	0.423	0.324	0.320
20	0.558(0.563)	0.333	0.347	0.441	0.320	0.338
24	0.568	0.357	0.356	0.465	0.357	0.362
28	0.569	0.341	0.350	0.453	0.333	0.346
32	0.553	0.318	0.333	0.451	0.332	0.326
36	0.567	0.331	0.343	0.506	0.328	0.329
40	0.568	0.390	0.380	0.469	0.373	0.389

In MEC environment, network bandwidth has a great impact on the performance of the offloading algorithm. In order to test the performance improvement of DRL-GMOM in comprehensive profit with the increase of bandwidth, the training data set with 12 task nodes is selected, and the model is retrained under different bandwidth, and the test set with 12 task nodes is used to test the model. Similarly, compared with the 5 algorithms mentioned above, it can be seen from the results shown in Figure 5 that with the increase of bandwidth, the performance of the DRL-GMOM is the largest under the same conditions. We can find that the performance of HEFT algorithm and greedy policy is better than Temporal-Difference learning when the bandwidth is about more than 13Mpbs. Because HEFT algorithm is actually a greedy policy, the greedy policy and the HEFT algorithm change roughly the same.

VI. Conclusion

In this paper, we design a DRL-based graph mapping offloading model, which is used to solve the task offloading problem with dependent subtasks in MEC environment. Firstly, We model the user's computing task as DAG, and then the DAG task is transformed into task vector sequence task by priority-topology-relative-entropy algorithm. Based on sequence to sequence model, we construct the graph mapping offloading model's actor network and critic network through GRU recurrent neural network. Finally, the model is trained by A3C deep reinforcement learning algorithm. By comparing the experimental results, we can conclude that the DRL-based graph mapping offloading model proposed in this paper can achieve higher user comprehensive profit, which proves its effectiveness and feasibility.

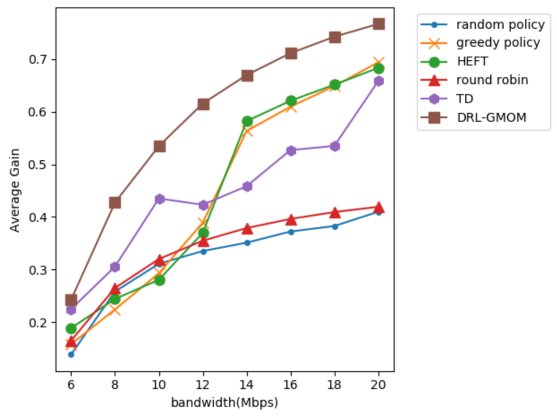

Fig. 5: Comparison of comprehensive profit of each algorithm with bandwidth increasing.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," IEEE Communications Surveys Tutorials, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, and D. Sabella, "On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration," IEEE Communications Surveys Tutorials, vol. 19, no. 3, pp. 1657–1681, 2017.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," IEEE Internet of Things Journal, vol. 3, no. 5, pp. 637–646, 2016.

[4] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: An overview," CoRR, vol. abs/1806.08894, 2018. [Online]. Available: http://arxiv.org/abs/1806.08894

[5] T. Yang, Y. Hu, M. C. Gursoy, A. Schmeink, and R. Mathar, "Deep reinforcement learning based resource allocation in low latency edge computing networks," in 2018 15th International Symposium on Wireless Communication Systems (ISWCS), 2018, pp. 1–5.

[6] L. Huang, S. Bi, and Y. J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," IEEE Transactions on Mobile Computing, vol. 19, no. 11, pp. 2581–2593, 2020.

[7] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," IEEE Transactions on Vehicular Technology, vol. 68, no. 11, pp. 11 158–11 168, 2019.

[8] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach," EURASIP J. Wirel. Commun. Netw., vol. 2020, no. 1, p. 188, 2020.

[9] L. Jin-zhong, X. Jie-wu, Z. Jin-tao, Z. Bing, and L. Chang-xin, “Survey on grid workflow scheduling algorithm,” Application Research of Computers, 2009.

[10] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, “Optimal joint scheduling and cloud offloading for mobile applications,” IEEE Transactions on Cloud Computing, vol. 7, no. 2, pp. 301–313, 2019.

[11] Maofei Deng, Hui Tian, and Bo Fan, “Fine-granularity based application offloading policy in cloud-enhanced small cell networks,” in 2016 IEEE International Conference on Communications Workshops (ICC), 2016, pp. 638–643.

[12] Z. Cheng, P. Li, J. Wang, and S. Guo, “Just-in-time code offloading for wearable computing,” IEEE Transactions on Emerging Topics in Computing, vol. 3, no. 1, pp. 74–83, 2015.

[13] C. Tsai and J. J. P. C. Rodrigues, “Metaheuristic scheduling for cloud: A survey,” IEEE Systems Journal, vol. 8, no. 1, pp. 279–291, 2014.

[14] Y. Zhang, Z. Zhou, Z. Shi, L. Meng, and Z. Zhang, “Online scheduling optimization for dag-based requests through reinforcement learning in collaboration edge networks,” IEEE Access, vol. 8, pp. 72 985–72 996, 2020.

[15] Z. Tang, J. Lou, F. Zhang, and W. Jia, “Dependent task offloading for multiple jobs in edge computing,” in 2020 29th International Conference on Computer Communications and Networks (ICCCN), 2020, pp. 1–9.

[16] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, “Fast adaptive task offloading in edge computing based on meta reinforcement learning,” IEEE Transactions on Parallel and Distributed Systems, vol. 32, no. 1, pp. 242–253, 2021.

[17] B. Barz, E. Rodner, Y. G. Garcia, and J. Denzler, “Detecting regions of maximal divergence for spatio-temporal anomaly detection,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 41, no. 5, pp. 1088–1101, 2019.

[18] 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 Communications Magazine, vol. 57, no. 5, pp. 64–69, 2019.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016, ser. JMLR Workshop and Conference Proceedings, vol. 48. JMLR.org, 2016, pp. 1928–1937.

[20] H. Arabnejad and J. G. Barbosa, “List scheduling algorithm for heterogeneous systems by an optimistic cost table,” IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 3, pp. 682–694, 2014.

[21] H. Topcuoglu, S. Hariri, and Min-You Wu, “Performance-effective and low-complexity task scheduling for heterogeneous computing,” IEEE Transactions on Parallel and Distributed Systems, vol. 13, no. 3, pp. 260–274, 2002.

[22] G. Otsuru and Y. Sanada, “Phase selection in round-robin scheduling sequence for distributed antenna system,” IEICE Transactions on Communications, 2020.