DQN-based Computation-Intensive Graph Task Offloading for Internet of Vehicles

Jinming Li[1], Bo Gu[1,2], Zhen Qin[1], Ziqi Lin[1] and Yu Han[1,2]
[1]School of Intelligent Systems Engineering, Sun Yat-sen University, Guangzhou 510275, China
[2]Guangdong Provincial Key Laboratory of Fire Science and
Intelligent Emergency Technology, Guangzhou 510006, China
{lijm89,qinzh23,linzq36}@mail2.sysu.edu.cn, {gubo, hanyu25}@mail.sysu.edu.cn

2022 IEEE Wireless Communications and Networking Conference (WCNC) | 978-1-6654-4266-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/WCNC51071.2022.9771951

***Abstract*—A computation-intensive graph task comprises a set of tasks and corresponding data flows between adjacent tasks. In this paper, we consider a mobile edge computing (MEC) system based on vehicle-to-everything (V2X) communication in which each task initiator (TI) generates and offloads a set of correlated tasks to different task executors (TEs). We formulate the graph task assignment problem as a mixed-integer nonlinear programming problem (MINLP) to minimize the weighted sum of time-energy consumption (WETC). Due to the data-flow dependency and time-varying characteristics of the operating environment including channel gain, communication distance between TIs and TEs, available computing resources of TEs, traditional numerical optimization algorithms cannot solve such optimization problem efficiently, especially when the scale of the MEC system is quite large. To this end, we propose a graph task offloading mechanism named GT-DQN by integrating deep Q-Network (DQN) with breadth-first search technique. Firstly, DQN is trained to generate a near-optimal offloading strategy, through numerous interactions with the time-varying operating environment. Secondly, a breadth-first search algorithm is adopted to traverse the graph task, which can significantly reducing the computational complexity. Compared with existing algorithms, simulation results verify the superiority of GT-DQN.**

***Index Terms*—Deep Reinforcement Learning, Graph Tasks, Mobile Edge Computing, Task Offloading, V2X**

I. Introduction

With the rapid development of V2X technology, intelligent connected vehicles (ICVs) [1] have come into reality, bringing immense benefits to our daily lives. It is estimated that 286 million ICVs will be manufactured in the world by 2025. Although ICVs nowadays are equipped with high performance computation devices, their on-board computational capabilities are still insufficient when executing computation-intensive tasks, e.g., the training of deep neural networks. Fortunately, mobile edge computing (MEC) [2], [3] addresses the above-mentioned issue by allowing ICVs to access nearby devices or edge servers to achieve more efficient data processing via task offloading.

Generally, task offloading for MEC is divided into two categories: binary offloading and partial offloading. In the first category, each task is either executed locally or offloaded to an edge server as a whole. In the second category, each task is divided into two parts that are executed locally and offloaded to an edge server, respectively. However, in the Internet of Vehicles (IoVs), some perception-related applications with computation-intensive features usually have multiple interdependent tasks, bringing challenges to task offloading.

Current research mainly focuses on modeling the interdependency between tasks as a directed task graph [4], [5], where tasks and data flows between tasks are represented by vertices and directed edges, respectively. Specifically, Zhang *et al.* [6] adopted partial critical path analysis to solve the general graph task scheduling problem. In [7], Hosseinalipour *et al.* proposed a distributed subgraph isomorphism extraction algorithm to find hidden excellent strategies when traversing geo-distributed cloud networks. In [8], LiWang *et al.* minimized task completion delay and data exchange cost through the graph task offloading algorithm based on subgraph isomorphism.

However, there are still some factors limiting the performance of these existing graph tasks offloading methods [6]–[8], such as the dynamics on wireless channel and fluctuations in available computing resources. Hence, a lot of research efforts have been devoted to strengthen the ability of anti environment disturbance for task offloading algorithms. In [9], the author solved the mixed integer optimization problem of graphics task offloading strategy through deep reinforcement learning (DRL). In [10], Yan *et al.* proposed a actor-critic framework to solve the joint optimization problem about offloading strategy and CPU freqencies. However, they only considered offloading tasks to a single device, and the computing power of the device is infinite, which is not in line with the reality.

In this paper, we firstly apply a deep Q-network (DQN) [11] to solve the problem of computation-intensive graph task offloading via V2X communication. Specially, we propose an offloading algorithm for ICVs to determine the graph tasks allocation in real time to minimize the weighted sum of time-energy consumption (WETC). on the premise of ensuring successful offloading. The main contributions are summarized as follows:

1) We formulate a mixed-integer nonlinear programming problem (MINLP) [12] to establish the offloading strat-

This work was supported in part by the National Science Foundation of China (NSFC) under Grand U20A20175 and in part by the National Key R&D Program of China under Grant 2020YFB1713800. *(Corresponding author:Yu Han.)*

egy of the graph task under time-varying conditions and limited resources (e.g., computing power) to minimize the transmission delay and energy consumption.
2) We propose a DQN-based graph task offloading mechanism via breadth-first search (GT-DQN) that fully takes into account the dependencies between tasks.
3) The proposed algorithm is based on DRL technology with free model and fast convergence, and can adapt to the dynamic changes of task topology and wireless environment.
4) The simulation results verify that our algorithm outperforms other schemes in dealing with intensive graph tasks of different topologies.

The remainder of this paper is organized as follows. Section II mainly introduces the system model, the cost model and the formulated optimization problem. Section III will present the basics of RL model based on the optimization problem. Section IV proposes a DQN based graph task offloading mechanism via breadth-first search. In Section V, simulation settings and results are described. Finally, we conclude the paper in Section VI.

II. System Model and Problem Formulation

A. System Model

As shown in Fig.1, each task initiator (TI) can interact with task executors (TEs), e.g., base stations (BSs) and ICVs, through one-hop V2X wireless links. For each slot, we consider a MEC system that is comprised of a single TI and multiple TEs $\boldsymbol{N} = \{1, 2, \ldots, N\}$. Each TE $n \in \boldsymbol{N}$ has a set of idle CPU cores $\boldsymbol{z_n}$ that can executes at most $|\boldsymbol{z_n}|$ inter-dependent tasks concurrently. Each CPU core can run one task with the same processing capacity on the same TE.

To conveniently reflect the different topological relationships of the graph tasks, the dependency of tasks is represented by an undirected graph $\boldsymbol{G} = \{\boldsymbol{M}, \boldsymbol{E}\}$, which contains a collection of tasks $\boldsymbol{M} = \{1, 2, \ldots, M\}$, where each task requires a CPU cores, along with a set of edges $\boldsymbol{E} = \{e_{mm'} \mid (m, m') \in \boldsymbol{M}, m \neq m'\}$. $|\boldsymbol{M}|$ represents the total number of tasks and each edge $e_{mm'} \in \boldsymbol{E}$ denotes a binary indicator variable indicating whether there is data exchange between m and m' or not. Fig.2 shows an example of a graph task containing nine inter-dependent tasks.

In addition, we assume that a dedicated spectrum resource block is allocated to each TE $n \in \boldsymbol{N}$ during the transmission process, which can support the concurrent transmission of task offloading and downloading.

B. Cost Model via V2X Communication

1) Transmission Delay: The transmission delay is the time required to transmit the computational task $m \in \boldsymbol{M}$ to TE $n \in \boldsymbol{N}$ in parallel. The transmission delay is calculated as follows:

$$t_{mn}^{\text{trans}} = \frac{l_m}{R_{mn}}, \forall m \in \boldsymbol{M}, n \in \boldsymbol{N} \tag{1}$$

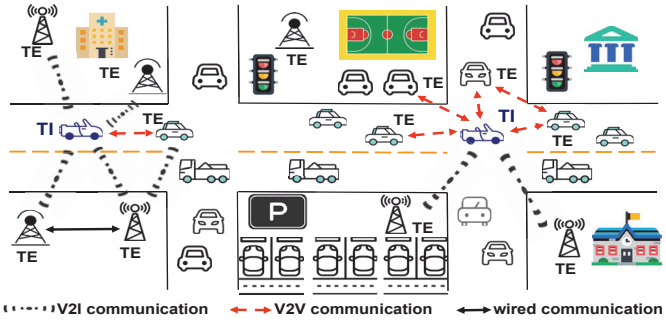

Fig. 1: Cooperative offloading framework.

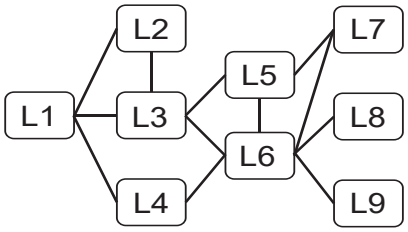

Fig. 2: The framework of graph task.

where l_m is the data size of computational task $m \in \boldsymbol{M}$ and R_{mn} denotes the uplink transmission rate, which can be obtained from Shannon's theorem.

$$R_{mn} = BW_n \log_2(1 + \frac{P_T h_n}{\sigma^2}) \tag{2}$$

where BW_n is the allocated bandwidth and h_n is the channel gain from TI to TE $n \in \boldsymbol{N}$. Besides, the power of uplink transmission is represented by P_T and the additive white Gaussian noise (AWGN) is denoted as σ^2.

Therefore, the total transmission time of TI offloading varying amounts of tasks to different TEs in parallel is equal to the maximum time among all transmission:

$$T^{\text{trans}}(u) = \max(\sum_{m \in \boldsymbol{M}} \sum_{n \in \boldsymbol{N}} u_{mn} t_{mn}^{\text{trans}}) \tag{3}$$

where u_{mn} denotes a binary indicator variable with $u_{mn} = 1$ if m is assigned to n and $u_{mn} = 0$ otherwise.

2) Execution Delay: When all tasks are transmitted to each TE $n \in \mathbf{N}$, cores in TE begin to execute each task in parallel. We consider that TEs have different speed in processing tasks due to the difference of CPU performance. Therefore, the total execution time of TEs executing varying amounts of tasks in parallel is:

$$T^{\text{exec}}(u) = \max(\sum_{m \in \boldsymbol{M}} \sum_{n \in \boldsymbol{N}} \frac{u_{mn} k_m}{f_n}) \tag{4}$$

where f_n represents the CPU frequency of TE $n \in \boldsymbol{N}$ and k_m indicates the computation workload of task $m \in \boldsymbol{M}$.

Consequently, we define the total completion delay as the sum of transmission delay and execution delay. Thus, the formulation is as follows:

$$T(u) = T^{\text{trans}}(u) + T^{\text{exec}}(u) \tag{5}$$

3) Data Exchange Cost: Supposing that there is no data exchange cost if two connected tasks are offloaded to the same TEs; however, if they are offloaded to different TEs, data will be transmitted between the both of TEs. Thus, the data exchange cost generated in the above process is denoted as $c_{nn'}((n,n') \in \boldsymbol{N}, n \neq n')$. This represents the cost of traffic exchange between different TEs in the MEC system. Therefore, we denote the consumption of data exchange between all TEs as follows:

$$E(b) = \frac{1}{2}\sum_{n\in\boldsymbol{N}}\sum_{n'\in\boldsymbol{N}} b_{nn'}c_{nn'} \tag{6}$$

where the binary indicator variable $b_{nn'}$ indicates whether there is data exchange between different TEs:

$$b_{nn'} = \begin{cases} 1 & n \neq n', u_{mn} * u_{m'n'} * e_{mm'} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

An example of data exchange consumption generated by offloading the graph task of Fig.2 is shown in Fig.3.

C. Problem Formulation

This paper takes WETC as a performance indicator, which can be defined as:

$$\tau = \omega T(u) + (1-\omega)E(b) \tag{8}$$

where $\omega \in [0,1]$ and $(1-\omega)$ represent the weight coefficients of completion delay and data exchange, respectively.

To achieve efficient offloading of general graph tasks, we formulate the following optimization problem with the central performance of minimizing WETC.

$$\begin{aligned} &\min_{\boldsymbol{z},\boldsymbol{f},\boldsymbol{d}} \ \tau \\ \text{s.t.} \ &(a) \quad u_{mn} \in \{0,1\} \quad \forall m \in \boldsymbol{M}, \forall n \in \boldsymbol{N} \\ &(b) \quad \sum_{m\in\boldsymbol{M}}\sum_{n\in\boldsymbol{N}} u_{mn} = |\boldsymbol{M}| \\ &(c) \quad \sum_{m\in\boldsymbol{M}} u_{mn} \leq |z_n| \qquad \forall n \in \boldsymbol{N} \\ &(d) \quad \sum_{n\in\boldsymbol{N}} u_{mn} \leq 1 \end{aligned} \tag{9}$$

where constraint (a) implies whether m is assigned to n or not.. Constraint (b) ensures that all tasks are allocated to idle cores. Constraint (c) ensures that the tasks offloaded to a TE cannot exceed its maximum computing resources. Constraint (d) limits that a task can only be offloaded to one TE at most.

It is worth noting that the above objective function is a MINLP problem which is NP-hard, and the reasons are as follows. First, the task offloading indicator u and the data exchange indicator b are both binary variables, resulting in a combinatorial problem. Second, considering that the constraint, the channel gain h and the optimization object are nonconvex, there exists a numerical locally optimal solution to the above problem [13], [14]. Therefore, we propose a DRL-based algorithm to solve the above problems with low computational complexity.

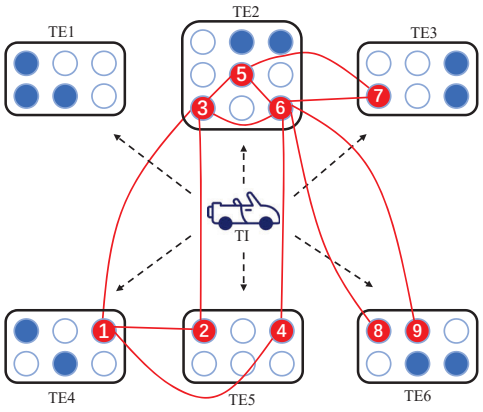

Fig. 3: Illustrative example for offloading a graph task of Fig.2, in which solid circles and hollow circles denote busy cores and idle cores, respectively. The cost of offloading the graph task is $E = 2c_{23} + c_{24} + 2c_{25} + 2c_{26} + 2c_{45}$.

III. DQN Based Graph Task Offloading Mechanism Via Breadth-First Search

A. Overview of Reinforcement Learning

In reinforcement learning (RL), the agent interacts with the environment and updates the learning strategies. Specifically, at time slot t, the agent determines an action a_t based on the current environment state s_t according to the policy $\pi_t(s,a)$. After receiving the action a_t, the environment transits to a new state s_{t+1} and generates a reward signal r_{t+1} fed back to the agent. Thus, the agent continuously interacts with the environment and generates a sequence of experience tuples $\{(s_t, a_t, r_{t+1}, s_{t+1}), \dots\}$. Based on those experience tuples, the agent updates its policy $\pi_t(s,a)$ to maximize the discounted return:

$$R_t = \sum_{k=0}^{+\infty} \gamma^k r_{t+k+1} \tag{10}$$

where $\gamma \in [0,1]$ denotes the discount rate.

As a value-based algorithm, the main idea of Q-learning is to update the Q-value $Q_\pi(s,a)$ by interacting with the environment, where $Q_\pi(s,a)$ is an utility of selecting action a in state s following policy π:

$$Q_\pi(s,a) = \mathbb{E}\left[R_t | s_t = s, a_t = a\right] \tag{11}$$

The optimal action-value function $Q^*(s,a) = \max_\pi Q_\pi(s,a)$ is the highest long-term reward for state s and action a. According to the Bellman equation, $Q^*(s,a)$ is given by:

$$Q^*(s,a) = \mathbb{E}_\pi[r_{t+1} + \gamma \max_\pi \ Q^*(s_{t+1}, a_{t+1}) | s_t = s, a_t = a] \tag{12}$$

The Q-learning algorithm maintains a Q-table with the Q-value $Q_\pi(s,a)$ as elements, and the agent updates the Q-value in the Q-table can be expressed as:

$$Q(s_t,a_t) \leftarrow (1-\alpha)Q(s_t,a_t)+\alpha(r_{t+1}+\gamma\max_{\pi}Q(s_{t+1},a_{t+1})) \tag{13}$$

where $\alpha \in [0,1]$ denotes the learning rate.

To avoid local optimum, the agent needs to explore some actions that have not been taken before to find a better potential strategy. Generally, we adopt the ϵ-greedy algorithm.

$$a=\begin{cases}\arg\max\limits_{a} Q(s,a) & \text{with probability } 1-\epsilon \\ \text{random} & \text{otherwise}\end{cases} \tag{14}$$

B. Setting of Reinforcement Learning

1) State: To ensure efficient offloading strategy, the state contains information about the communication environment. In summary, state features observed by agent are denoted as:

1) $\boldsymbol{h}=\{h_n\}_{n\in\boldsymbol{N}}$: Channel gain of TE $n\in\boldsymbol{N}$.
2) $\boldsymbol{f}=\{f_n\}_{n\in\boldsymbol{N}}$: CPU frequency of TE $n\in\boldsymbol{N}$.
3) $\boldsymbol{z}=\{z_n\}_{n\in\boldsymbol{N}}$: Idle cores of TE $n\in\boldsymbol{N}$.
4) $\boldsymbol{d}=\{d_n\}_{n\in\boldsymbol{N}}$: Distance between TE $n\in\boldsymbol{N}$ and TI.
5) $\boldsymbol{G}$*:* The topology of current graph task to be offloaded.

Therefore, the combined state of the system can be denoted as $\boldsymbol{s}=\{\boldsymbol{h},\boldsymbol{f},\boldsymbol{z},\boldsymbol{d},\boldsymbol{G}\}$.

2) Action: Observing the state, the agent will select the suitable TE to offload according to the attachment relationship between the current task $m\in\boldsymbol{M}$ and other tasks in the graph task. Thus, we express the action set $\boldsymbol{a}$ as:

$$\boldsymbol{a}=\{a_{m,n}|\ m\in\boldsymbol{M},n\in\boldsymbol{N}\} \tag{15}$$

where $a_{m,n}=1$ if choosing TE $n\in\boldsymbol{N}$ to offload task $m\in\boldsymbol{M}$ and $a_{m,n}=0$ otherwise.

3) Reward: Combining with the system model, we elaborate a reward function which aims to minimize the WETC:

$$r=\frac{1}{\omega T(u)+(1-\omega)E(b)} \tag{16}$$

In this way, the agent can learn the near-optimal offloading strategy from the interactions with its operating environment.

C. Deep Q-Learning Network

According to the settings of RL, the proposed system has a sizeable state-action set. Traditional RL-based algorithms are costly to maintain a very large lookup table in memory and most of the states are rarely accessed.

On this basis, DQN shows excellent performance in solving dynamic multi-dimensional MDP problems, which is one of the essential innovations of DRL. Compared with Q-learning, DQN has made further improvements in three aspects: firstly, the convolutional neural network is used to approximate value function; secondly, target Q-Network makes the training of neural network stable; thirdly, experience replay is used to solve correlation and non-static distribution problems. Specifically, DQN uses a main network to generate current Q-value $Q(s,a;\boldsymbol{\theta})$, a target network to evaluate the value function $Q(s,a;\boldsymbol{\theta'})$ of current state-action pair. To update the parameters of the Q-network, we define the loss function as the mean square deviation between the current Q-value and the target Q-value.

$$\text{loss}(\boldsymbol{\theta})=\mathbb{E}\left[(r_{t+1}+\gamma\max_{\pi}\ Q(s_{t+1},a_{t+1};\boldsymbol{\theta})-Q(s_t,a_t;\boldsymbol{\theta}))^2\right] \tag{17}$$

where $\boldsymbol{\theta}$ means a set of parameters in the Q-network, the terms in the square is temporal-difference error. In the process of updating, neural network updates the parameter $\boldsymbol{\theta}$ by:

$$\boldsymbol{\theta}=\boldsymbol{\theta}+\beta\nabla\mathcal{L}(\boldsymbol{\theta}) \tag{18}$$

where β represents the learning rate, $\nabla\mathcal{L}(\boldsymbol{\theta})$ represents the first-order partial derivative of $\boldsymbol{\theta}$.

Algorithm 1 Training process of GT-DQN

Initialize:
Initialize $Q(s,a;\boldsymbol{\theta})$ with weight $\boldsymbol{\theta}$.
The weight of target network $\boldsymbol{\theta'}=\boldsymbol{\theta}$.
Initialize replay buffer $\mathcal{D}$.
for episode $=1,2,\cdots,W$ **do**
Receive initial observation state.
Set up an empty queue. $i\leftarrow 1$, idle state $\leftarrow$ true.
while $i\le|\boldsymbol{M}|$ and idle state **do**
if there is no idle core **then**
idle state $\leftarrow$ false.
Exit the cycle.
end if
if $i=1$ **then**
Randomly select one task as the head node.
end if
Taking out the task $m\in\boldsymbol{M}$ to be allocated from the queue in turn and add i by one.
Queue all the tasks connected with the task $m\in\boldsymbol{M}$ in turn according to the graph task $\boldsymbol{G}$.
Select action a_i based on the probability ϵ according to Eq.(14) and queue all the tasks connected with the task.
Observe reward r_i and next state s_{i+1}.
Store transition (s_i,a_i,r_i,s_{i+1}) in $\mathcal{D}$.
Sample a random mini-batch of transitions from $\mathcal{D}$ when $\mathcal{D}$ is full.
Update DQN using the stochastic gradient descent.
Update the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta'}$ using Eq.(18).
end while
end for

D. Proposed GT-DQN Algorithm

Algorithm 1 shows the training process of GT-DQN. Since the queue is in the order of first in first out, we use the queue data structure to realize breadth-first search. An empty queue

TABLE I: Simulation Parameters

Parameters	Values
Bandwidth (e.g.,BW_n)	1 MHz
Noise power (σ^2)	10^{-10} W
Antenna gain (A_d)	4.11
Carrier frequency (f_c)	915 MHz
Pass loss exponent (LE)	3
Data exchange cost (e.g.,$c_{nn'}$)	0.1
Transmit power (P_T)	2 W
Learning-rate (α)	0.001
Distance between TI and TE (e.g.,d_n)	$<$ 200 m
Data size of each task (e.g.,l_m)	[50, 150] KByte
Computing workload (e.g.,k_m)	[50, 150] Mcycles
CPU frequency (e.g.,f_n)	[20, 50] GHz

should be set up before TI offloads the graph task. At the initial stage, the agent randomly selects one task as the head node and receives initial observation state. Then, the agent takes out the task $m \in \boldsymbol{M}$ to be allocated from the queue in turn.

In decision epoch i (i.e., when offloading i-th task), the agent selects an action a_i based on Eq.(14) according to the current environment state s_i, and stores all the connected tasks into the queue in turn to achieve breadth-first search according to the graph task $\boldsymbol{G}$. The agent receives the feedback signal from the environment, consisting of an instantaneous reward r_i and the next state s_{i+1}. Then, the transition tuple (s_i, a_i, r_i, s_{i+1}) is stored as an experience into the experience replay memory buffer $\mathcal{D}$. When the replay memory buffer is full, the experience is randomly selected to generate mini-batches from it to train DQN. Agent repeats the above steps until offloading is completed.

Through the breadth-first search method, we only need to visit each node in the graph task once according to the generated queue and choose the unique idle core to offload through the GT-DQN algorithm. Therefore, the computational complexity of each iteration is only $O(M)$, which significantly improves the performance of the running time.

IV. Numerical Evaluation

In this section, we evaluate the performance of GT-DQN and analyze the experimental results.

A. Experiment Settings

As shown in Fig.4, we consider five different graph tasks, each of which consists of different tasks corresponding to different topological relations. The channel gain is generated with Rayleigh distribution and each channel follows the Rayleigh fading model. Since WETC cannot be calculated when offloading fails (i.e., $\exists n \in \boldsymbol{N}, \sum_{m \in \boldsymbol{M}} u_{mn} > |z_n|$), we set a maximum threshold to represent the penalty for failure. Assuming that TIs in the MEC system always suffer the pressure of insufficient computing resources, so they are more inclined to offload tasks to TEs. Other parameters are shown in Table I.

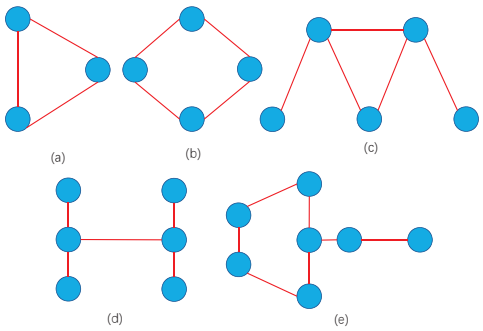

Fig. 4: Different graph task topologies corresponding to different number of tasks.(a) closed triad graph; (b) square graph; (c) bull graph; (d) double-star graph; (e) tadpole graph.

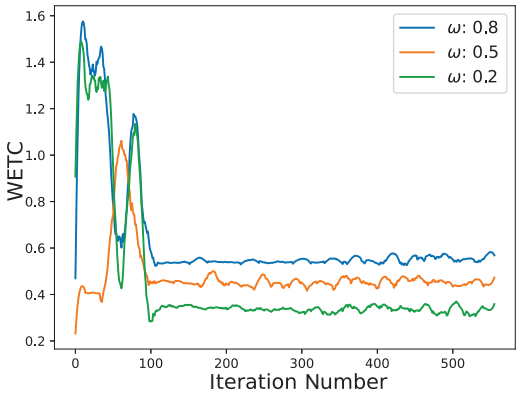

Fig. 5: Performance comparison of WETC with different weight coefficients (N = 4).

B. Evaluation of Convergence

For the sake of generality, we test the convergence of GTDQN with different weight coefficients. Assuming that there are five tasks offloaded to four TEs. As shown in Fig. 5, we assess the impact of different weight coefficients on the convergence and results show that the convergence of GTDQN is relatively stable. In view that the cost of transmission time is greater than the cost of data exchange, we can clearly observe that when the weight coefficient of transmission time is smaller, the WETC generated by offloading is obviously lower. When the weight coefficient of transmission time becomes smaller, agents tend to offload the task to the same TE.

C. Performances of GT-DQN

Considering different number of TEs, Fig.6 depicts the performance comparison of WETC generated by offloading the different graph task types in Fig.4. The number of TEs in subgraphs (a) and (b) are 4 and 6, respectively. The number of tasks and associated edges in the graph task reflect the intensive degree of task computing. The larger the nunber of tasks and associated edges is, the higher WETC is achieved. At the same time, due to the randomness of idle cores, the probability of offloading failure also increases, which puts forward more significant requirements for the feasibility of

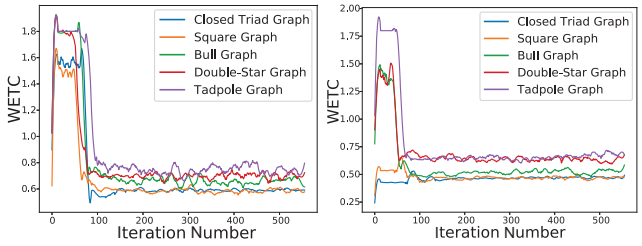

Fig. 6: Performance comparison of WETC with different number of TEs ($\omega = 0.5$). (a) $N = 4$. (b) $N = 6$.

GT-DQN. When the number of TEs is four, the offloading results fluctuate to a certain extent, which is due to the shortage of equipment needed for distribution and the instability. With the number of TEs increases, the convergence rate increases steadily, and the offloading performance also becomes stable. Therefore, these also prove that our algorithm is in line with the actual situation.

D. Comparison with Existing Algorithms

We also implement the performance comparison of GT-DQN with three existing algorithm.

1) *R-HTSI:* The random algorithm via hierarchical tree-based subgraph isomorphism [8].
2) *R-HTSI-W:* On the basis of the R-HTSI algorithm, the unassigned tasks are placed in the waiting sequence until the emergence of idle CPU cores.
3) *Exhaustive Search:* The optimal offloading strategy can be obtained through exhaustive search. Although exhaustive search is infeasible due to high cost, it servers as a benchmark for evaluating the performance.

The graph task to be offloaded is shown in Fig.4 (e). As Fig.7 shown, we can find that the probability of offloading failure increases when using R-HTSI to offload graph task in an unknown environment, and cause frequently the penalty for failure. Therefore, WETC of R-HTSI is greater than that of the other three algorithms. In addition, R-HTSI-W still generates high completion delay although there is no failure penalty. In summary, the comprehensive performance of GT-DQN is superior to that of the other three algorithms for offloading a computing-intensive graph task.

V. Conclusion

In this paper, we mainly proposed a DQN based graph task offloading mechanism via breadth-first search. By combining DQN and breadth-first search, we solved the combinatorial optimization problem caused by the strong coupling between tasks, and effectively reduced the time complexity of task offloading. Moreover, a significant feature of our proposed algorithm is that it can guide agent to minimize WETC by a series of state-action-reward observations while interacting with the time-varying operating environment. We compared the proposed algorithm with other existing algorithms. The

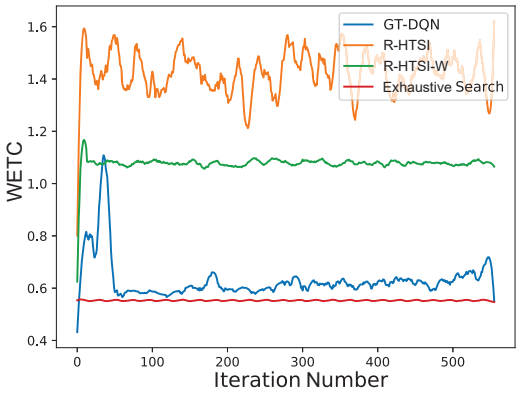

Fig. 7: Comparison with different algorithms ($N = 4$, $\omega = 0.8$).

simulation results showed that our algorithm has good convergence in different scenarios and can obtain near-optimal solutions.

References

[1] H. Wu, J. Zhang, Z. Cai, F. Liu, Y. Li, and A. Liu, "Toward energy-aware caching for intelligent connected vehicles," *IEEE Internet of Things Journal*, vol. 7, no. 9, pp. 8157–8166, 2020.
[2] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.
[3] B. Gu, X. Zhang, Z. Lin, and M. Alazab, "Deep multiagent reinforcement-learning-based resource allocation for internet of controllable things," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3066–3074, 2021.
[4] K. Zhang, J. Cao, and Y. Zhang, "Adaptive digital twin and multiagent deep reinforcement learning for vehicular edge computing and networks," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2021.
[5] Y. Yao, B. Gu, Z. Su, and M. Guizani, "Mvstgn: A multi-view spatial-temporal graph network for cellular traffic prediction," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.
[6] W. Zhang and Y. Wen, "Energy-efficient task execution for application as a general topology in mobile cloud computing," *IEEE Transactions on Cloud Computing*, vol. 6, no. 3, pp. 708–719, 2018.
[7] S. Hosseinalipour, A. Nayak, and H. Dai, "Power-aware allocation of graph jobs in geo-distributed cloud networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 749–765, 2020.
[8] M. LiWang, S. Hosseinalipour, Z. Gao *et al.*, "Allocation of computation-intensive graph jobs over vehicular clouds in iov," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 311–324, 2020.
[9] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5404–5419, 2020.
[10] J. Yan, S. Bi, L. Huang, and Y.-J. A. Zhang, "Deep reinforcement learning based offloading for mobile edge computing with general task graph," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–7.
[11] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, "Playing atari with deep reinforcement learning," 2013.
[12] C. Floudas, P. Pardalos, C. Adjiman, W. Esposito, Z. Gumus, S. Harding, J. Klepeis, C. Meyer, and C. Schweiger, *Mixed-Integer Nonlinear Programming Problems (MINLPs)*, 01 1999, pp. 263–302.
[13] M. Peng, Y. Yu, H. Xiang, and H. V. Poor, "Energy-efficient resource allocation optimization for multimedia heterogeneous cloud radio access networks," *IEEE Transactions on Multimedia*, vol. 18, no. 5, pp. 879–892, 2016.
[14] B. Gu, X. Yang, Z. Lin, W. Hu, M. Alazab, and R. Kharel, "Multiagent actor-critic network-based incentive mechanism for mobile crowdsensing in industrial systems," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 9, pp. 6182–6191, 2021.