A Socially-Aware Dependent Tasks Offloading Strategy in Mobile Edge Computing

Yanqi Gong, Fei Hao, Liang Wang, *Member, IEEE*, Liang Zhao, *Member, IEEE*, and Geyong Min, *Member, IEEE*

***Abstract*—With the advent of 5G, Mobile Edge Computing (MEC), a promising computing paradigm sits closer to users than cloud computing, is being broadly used in various Internet of Things (IoT) applications, and achieve high-quality user experience. Task offloading, as a critical research issue in MEC, is playing an important role in optimizing computational resources and management. However, many tasks are executed dependent on the computational results of other tasks. Moreover, in the case of offloading tasks with other devices, it is often required to consider the success rate of offloading, since not all users are willing to lend their mobile devices to others for task execution. To address this challenge, by taking social relationships between users into account, this paper intends to combine computational resources of local devices and edge clouds and provide more flexible offloading and execution solutions, for achieving the efficient offloading of dependent tasks with the joint consideration of network latency and energy consumption. This paper develops a dependent task offloading strategy based on Bipartite Graph Matching. Extensive simulations are conducted for validating the effectiveness of our proposed strategy. Experimental results demonstrate that our proposed strategy can significantly minimize the overhead compared with other baseline strategies. In particular, the overhead is reduced 8.2%, compared with the strategy which consider the Device-to-Device (D2D) offloading only.**

***Index Terms*—D2D, dependency task offloading, energy consumption, mobile edge computing, network latency.**

Manuscript received 20 April 2022; revised 30 December 2022; accepted 23 January 2023. Date of publication 30 January 2023; date of current version 8 September 2023. This work was supported in part by European Union's Horizon 2020 research and innovation programme through the Marie Sklodowska-Curie under Grant 840922, in part by the National Natural Science Foundation of China under Grants 61702317 and 62071283, in part by the Ministry of Education Humanities and Social Sciences Research Youth Fund Project under Grant 22YJCZH046, in part by the Natural Science Basic Research Plan in Shaanxi Province of China under Grant 2022JM-371, and in part by the Fundamental Research Funds for the Central Universities, China under Grants GK202103080 and GK202003075. Recommended for acceptance by M. Obaidat. *(Corresponding authors: Fei Hao, Liang Zhao.)*

Yanqi Gong and Liang Wang are with the Key Laboratory of Modern Teaching Technology, Ministry of Education, Xi'an 710062, China, and also with the School of Computer Science, Shaanxi Normal University, Xi'an 710119, China (e-mail: gyqi@snnu.edu.cn; wangliang@snnu.edu.cn).

Fei Hao is with the Key Laboratory of Modern Teaching Technology, Ministry of Education, Xi'an 710049, China, and also with the School of Computer Science, Shaanxi Normal University, Xi'an 710119, China, and also with the Department of Computer Science, University of Exeter, EX4 4QF Exeter, U.K. (e-mail: feehao@gmail.com).

Liang Zhao is with the School of Computer Science, Shenyang Aerospace University, Shenyang 110136, China (e-mail: lzhao@sau.edu.cn).

Geyong Min is with the Department of Computer Science, University of Exeter, Exeter EX4 4QF, U.K. (e-mail: g.min@exeter.ac.uk).

Digital Object Identifier 10.1109/TSUSC.2023.3240457

I. Introduction

In recent years, the fifth generation (5G) of mobile communication system has been born with the continuous development of mobile technology, the connection of massive smart devices, and the booming growth of mobile data volume [1], [2]. With the arrival of 5G, applications such as Internet of Things, virtual reality, and ultra-high-definition video are all being used in people's daily lives widely [3]. However, these applications often require high bandwidth and low latency, in which Mobile Edge Computing (MEC) has emerged in response to this demand. Without MEC, all aforementioned applications were executed on mobile devices or cloud platforms. If we process the computing on mobile devices, the computing resources of mobile devices are too small [4]. In contrast, if we process them on cloud, the resource-intensive applications often requires transmitting huge amounts of data between mobile device and remote cloud server, resulting in unpredictable communication latency [5], [6]. Therefore, Mobile Edge Computing (MEC) becomes a promising solution to overcome these drawbacks [7], [8], [9], [10], [11]. Compared with cloud computing, MEC server is a device sitting closer to the user with good computational process ability. Thus, MEC can provide a better service experience with lower latency.

However, there are still many challenges with MEC. In real life, many tasks do not exist alone, in which it may need to be interdependent with other tasks, i.e., task requests from the same user are often be divided into a set of independent or dependent tasks, which are denoted as jobs [12]. For example, data compression, transmission, and display, these dependent tasks associated with the same job need to be executed sequentially according to the dependency relationships, i.e., the subsequent tasks need to wait for all their predecessors to complete their execution. Dependency relationships between tasks are critical factors that require to be considered for offloading decision-making [12], [13].

Take the example of collaborative traffic real-time monitoring in social awareness, which happens all around us. First of all, the purpose of socially aware collaborative traffic real-time monitoring is to collect traffic conditions such as congestion or accidents in a timely manner. Second, such applications are useful for many transportation services such as route planning, traffic management [14]. Moreover, the execution of a task with traffic condition information is generally dependent on the other tasks. This is because only after executing the previous task with

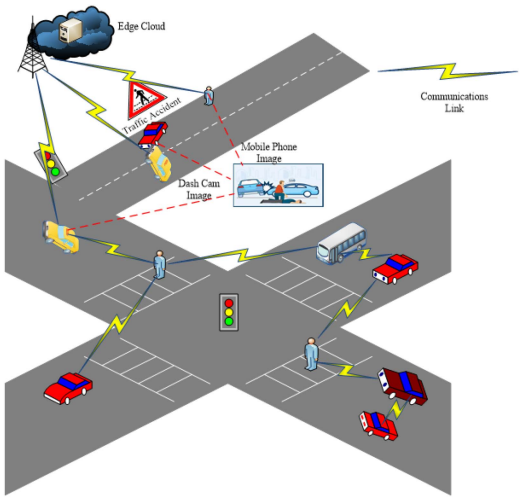


Fig. 1. Collaborative traffic real-time monitoring application.

traffic condition information, the current task can make the next decision based on the results. Traditionally, traffic monitoring is performed by analyzing the data from statically installed traffic cameras, which generally do not have a dense coverage [15]. And the data collected by these traffic cameras is required to be transmitted to the cloud for analysis and processing, which invariably brings relatively high bandwidth, energy and latency costs.

A socially-aware dependent tasks offloading strategy in MEC can be a good solution for addressing the above challenge. When we consider social relationships, we can choose the target device to offload among the devices that have social relations with the users. Hence, the range of target devices to choose from will be narrowed, thereby both latency and energy consumption can be reduced. In particular, the vehicles and pedestrians equipped with sensing devices (e.g., mobile phones, GPS sensors, dash cam) provide an opportunity for collecting a large amount of traffic data in real-time. For example, a networked dash cam can capture traffic conditions in front of the car and then transmit the data to an edge server for further analysis. In addition, pedestrians in the car can use their mobile phones to react in real time to the traffic conditions on their road. Pedestrians and drivers on other roads can share information via their mobile phones, as shown in Fig. 1. This allows drivers to access traffic conditions in real time through edge servers and others. The relevant authorities can also deal with the problematic road sections and control the traffic accordingly. There have been some relevant research works for addressing the problem in the above scenario. Cuervo et al. proposed an energy-efficient task offloading framework called MAUI in [16], to fully utilize the computational resources of users' friends. Some researchers attempted to incorporate the social relationships into the task offloading in MEC. Chen et al. [17] devised an unique socially-motivated cooperative MEC paradigm, in which the social connection structure among mobile and wearable device users are used to perform effective collaborative computing task executions. In Nonorthogonal multiple access (NOMA)-assisted energy-harvesting large IoT applications, Pei suggested a socially conscious joint resource allocation and compute offloading [18].

However, the above studies either investigated the dependent task offloading without taking social awareness into account, or studied the socially-aware offloading for the tasks without dependencies, which will both affect the success rate of task offloading to a certain extent. In fact, the social awareness can be used to optimize the dependent task offloading in MEC and further facilitate the computation of various IoT applications. Therefore, this paper focuses on the offloading between tasks with dependency relationship. To be specific, we consider both network latency and energy consumption and develop an optimized dependent tasks offloading strategy in MEC. Considering the user's requirements and real-time device resources, we adopt the following four types of offloading strategies, (1) local execution on the devices, (2) execution locally by virtue of Device-to-Device (D2D) offloading, (3) execution on the edge clouds directly, and (4) execution on the edge clouds for D2D offloading. In addition, we take social awareness into account as well, transform the social relationships of users in practice into the social relationships of devices, in order to improve the effectiveness and efficiency of tasks offloading. It is worth mentioning that, in the case of offloading tasks with other devices, it is often required to consider the success rate of offloading, since not all users are willing to lend their mobile devices to others for task execution. To address this challenge, by taking social relationships between users into account, this paper proposes a socially-aware dependent tasks offloading strategy. Therefore, how to obtain the social relations between users is a very challenging problem. In addition to this, this paper adds social awareness to the constraints, making the optimization model more complex and harder to implement. The major contributions of this paper are summarized as follows.

- The problem on socially-aware dependent task offloading addressed in this paper is modeled as a multi-objective optimization problem. Technically, we constrain tasks in terms of latency to satisfy dependencies between tasks. With this constraint, we define the overhead function U by taking both network latency and energy consumption into account. Further, an optimized offloading strategy of dependent tasks is obtained by minimizing the U value.
- The vertex set in bipartite graph can be partitioned into two mutually disjoint subsets, and the two vertices dependent on each edge in the graph belong to these two mutually disjoint subsets, and the vertices within the two subsets are not adjacent to each other. Inspired by these unique features of bipartite graph, the set of tasks and set of devices can be regarded as two vertex sets. Hence, it is very convenient and effective to model the problem of dependent tasks offloading with the bipartite graph matching. And we integrate the social relationships into the process of device computation and network resource sharing. Therefore, the selection range of target devices is narrowed, thereby both latency and energy consumption can be reduced. Hence, we propose a socially-aware dependent tasks offloading method based on bipartite matching algorithm.

- Extensive simulation are conducted for validating the effectiveness of our proposed socially-aware dependent tasks offloading strategy in terms of network latency and energy consumption. The simulation results demonstrate that our proposed algorithm can significantly save the energy consumption and reduce the network latency. The reason is that we consider both latency and energy consumption, so that the final selected offloading strategy will lead to a minimal overhead U, and further improve user's service experience. Furthermore, we present the 95% confidence interval of the results to indicate that the difference in the results is not due to the randomness of the data.

The structure of this paper is organized as follows. Section II presents an overview of related work. After constructing a system model of our proposed socially-aware dependent tasks offloading scheme, the problem formulation of this research is provided in Section III. Section IV describes the offloading strategy for the proposed problem and describes the algorithm, and then simulation results and performance evaluation are presented in Section V. Finally, Section VI concludes this paper including future work.

II. Related Work

MEC has received a lot of attentions from both academia and industrial in recent years [19], [20], [21]. Recently, the European Telecommunications Standards Institute (ETSI) has introduced the idea of Multi-Access Edge Computing (MAEC) in their 5G standard [22]. The distributed MEC servers at the network's edge are used in MAEC to deliver cloud computation and IT services with low latency, high bandwidth, and real-time processing. Task offloading is one of the most important aspects of MEC [7], [23]. The majority of prior research has focused on reducing the computational delay of tasks, improving service quality, or lowering the energy consumption of edge devices [24], [25], [26].

Mao et al. [27] investigated the problem of dynamic computation offloading for single-device with energy harvesting capability in MEC. Cuervo et al. [16] addressed the joint optimization problem of task offloading decision and resource allocation with the aim of reducing the total energy consumption of the system. In particular, Chen et al. [28] presented an energy-efficient task offloading approach that aims to reduce the total energy consumption of collaborative task execution amongst devices. Reference [29] proposes an energy-efficient D2D-assisted relay framework to minimize signaling overhead and energy consumption. Combining edge computing and D2D communication technology, Yang [30] proposed a joint optimization scheme of task offloading and resource allocation based on edge computing in 5G communication network to improve task processing efficiency. The joint computational offloading and user association problem was addressed in Ref. [31] in order to reduce the overall energy consumption of both mobile users and MEC servers. Chen et al. [8] investigated the task offloading problem to minimize the latency while saving the battery life of the device. [32] studied the multi-user multi-task computational offloading problem, where they determined the energy harvesting strategy and task offloading decision by using the Lyapunov optimization approach. In [33], a novel task offloading problem was studied, in which each task demands a specific network function with a tolerable delay, with the goal of increasing the number of authorised requests while lowering the operating cost. Green MEC with an energy harvesting device was studied in [27], where an efficient computational offloading strategy was devised to reduce execution delays and task failures. The task offloading problem of completing computational tasks under a given delay constraint while minimizing the total communication and computational energy consumption was investigated in [34]. Chen et al. [35] proposed a strategic computation offloading algorithm based on double deep Q-network (DQN) to learn optimal strategies without prior knowledge of network dynamics.

You et al. [20] presented an energy-efficient computational resource allocation paradigm for for MEC with the assumption that the device users are prone to cooperate. The social factors are emerging as a novel and important dimension in the design of future network systems. For example, Zhao et al. [36] presented a social computing-assisted data packet forwarding scheme in Vehicular Ad-hoc Networks (VANETs). Regarding tasks offloading in IoT applications, Pei et al. [18] developed a socially aware joint resource allocation and computation offloading strategy for reducing the energy consumption as well as network latency. The reference [37] studies the problem of component allocation considering social relationships in multi-access edge computing. An energy-efficient task assignment method based on Monte-Carlo search tree (MCTS), named task assignment solution based on Monte Carlo Tree Search algorithm (TA-MCTS) is proposed. Fan et al. [38] developed a collaborative task offloading scheme for D2D-assisted fog computing networks, in order to maximize the social revenue of each user, and reduce the average task execution latency.

The dependencies of tasks are important factor to be considered since the applications may not be executed successfully if the dependencies of tasks are not taken into account when offloading these applications [39]. Existing work on service caching has typically focused on the problem of joint optimization of services deployment and tasks offloading in MEC [4], [40], [41], [42]. Lin et al. [43] proposed a D2D (Device-to-Device) collaborative computation offloading and resource allocation system to minimize the task execution cost. Aiming to maximize the computation rate of all end devices, [44] investigated the joint optimization problem of computational paradigm selection and system transmission time allocation for end devices by considering a binary offloading model. In MEC, Deep Reinforcement Learning (DRL) is used to deal with the task offloading problem [45], [46], [47], [48]. However, these methods assume that offloading tasks are independent without considering the intrinsic dependencies between tasks in real applications. In practice, many applications are composed of related tasks, where the output of some tasks are the inputs of other tasks. A Maximum Reliability Offloading Algorithm (MROA) is proposed [49]. The main idea is to decompose the given constraints and dynamically adjust the decomposed constraints. The reference [50] establishes an actor-critic mechanism embedded in two layers for the DAG-based multi-tasks

TABLE I
SUMMARY OF EXISTING WORK

Scheme	Dependent	Latency	Energy	Socially-aware
[29], [30], [52]	×	✓	✓	×
[36], [38]	×	✓	×	✓
[18], [37]	×	✓	✓	✓
[50], [51]	✓	✓	✓	×
Our Scheme	✓	✓	✓	✓

computing offloading strategy in the MEC system, and proposes a DRL-based algorithm. Recently, Lv et al. [51] proposed a table-based task offloading algorithm (TBTOA). While satisfying service cache constraints and UE battery capacity constraints, TBTOA jointly optimizes the task completion time and energy consumption of edge servers. Although the above works consider the dependencies between tasks, they do not consider social relationships.

Table I summarizes the features of existing work.

Although all aforementioned works have studied task offloading in MEC, most of them only considered energy consumption or latency separately. The newly existing works investigated the task offloading model with the consideration of both latency and energy consumption, but they have not taken social awareness or the dependencies between tasks into account. It may cause the failure of task offloading or inaccurate execution results. Motivated by this, our work takes into account not only the dependencies between tasks, but also the social relationships, with the joint optimization of network latency and energy consumption, that makes our task offloading model more comprehensive in this work.

III. System Model and Problem Formulation

This section discusses the system model which can provide a good understanding to the structure of the whole system. Our system model is composed of task dependency model and network model. To be specific, the task dependency model describes how the dependencies between tasks are represented, and the network model characterizes the offloading of tasks in the edge nodes. Then, the problem is also formulated.

A. Task Dependency Model

We assume that one or several applications (e.g., vehicle recognition, face recognition) got to be executed at the edge of network. The execution of those applications is usually divided into multiple tasks, each of which may be offloaded and executed at one edge node. The set of tasks can be represented by $V=\{v_1, v_2, \ldots, v_n\}$ and $n = |V|$ denotes the amount of tasks. Considering the dependencies between the tasks, a Directed Acyclic Graph (DAG) $G = (V, E)$ representing the dependencies between tasks with V indicating the set of tasks and E denoting the set of edges representing precedence constraints, is employed for modelling the dependencies between the tasks. Mathematically, an edge exists between task v and task v' *iff* there is data transmission between them (i.e., the task v' can be started execution only when task v completed and sent the corresponding execution results to task v'). Thus, we use $a_{vv'}$ to denote the amount of data that needs to be transmitted from task v to task v'.

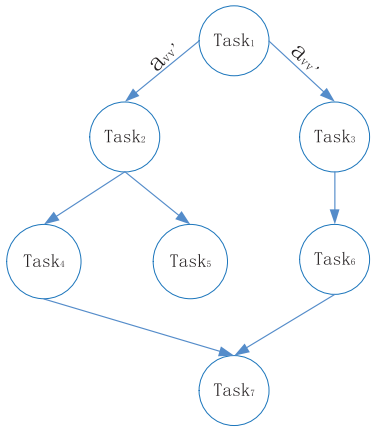


Fig. 2. Dependencies between tasks.

As shown in Fig. 2, we assume that the set of tasks $n = 7$, and there is a dependency between task 1 and task 2. Obviously, task 2 can only be executed after receiving the execution results of task 1.

B. Network Model

In our network model, the set of edge nodes is denoted as $M = \{m_1, m_2, \ldots, m_l\}$ with $l = |M|$ indicating the number of edge nodes which are connected each other through various network connections. More, the unit data communication latency from edge node m to m' is denoted by $c_{mm'}$ (if $m = m'$, then $c_{mm'} = 0$). Suppose M_v denote the set of edge nodes satisfying the service constraints of task $v \in V$. This implies that task v can only be executed by one edge node in M_v. Due to resources constraint, we assume that each edge node $m \in M$ dedicates just $C(m)$ CPU cycles to these tasks. Task v will take t_{vm} execution time and r_{vm} CPU cycles per second if it is offloaded to edge node m.

Fig. 3 shows a mapping between tasks and edge nodes. It depicts that the set of edge nodes that satisfy the tasks. Concretely, task 1 can only be executed at edge node m_1 and task 2 can only be executed at edge node m_2.

C. Social Relationship Model

When the two devices belong to the same user, or the users of the two devices have a certain social relationship, such as a kinship, a friendship, and a colleague relationship. In practical applications, two device users can quickly detect their social relationships by identifying the common social characteristics between them through a local matching process. For example, two device users can detect the social relationship between them by visiting online social media such as WeChat and Twitter. The social relationship between users affects the degree of trust users have in others. In this paper, we assume that the user has

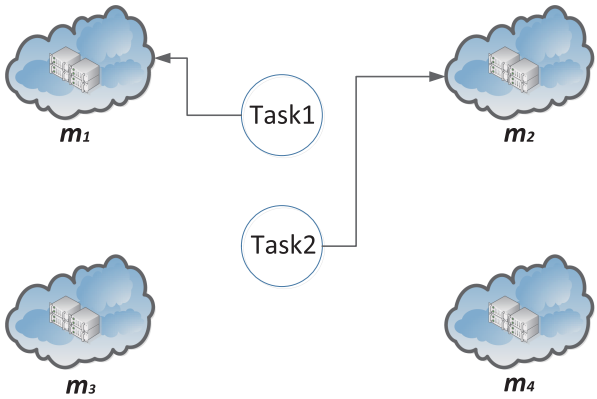


Fig. 3. Mapping between tasks and edge nodes.

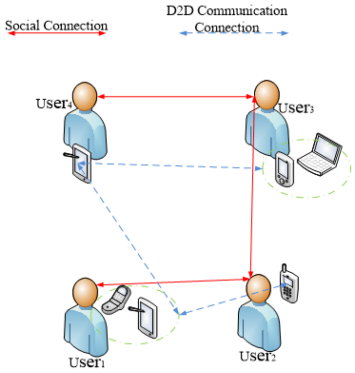


Fig. 4. Social relationship model.

trustworthy social relationships with his friends and relatives. When a task issued by a user needs to be offloaded to other mobile devices for execution, we first consider the mobile device that has a social relationship with the user. However, not all mobile devices that have a social relationship with the user are eligible for offloading. When the distance between the user and the mobile device of his relatives and friends is long, the user cannot offload tasks to perform on the mobile device. Therefore, the target device for task offloading needs to satisfy two conditions. First, there is a social relationship between the user device and the target device. Second, the transmission distance between the user device and the target device is relatively short, that is, there is a connection relationship between the devices.

As shown in Fig. 4, we assume that $User_1$ trusts $User_2$ since $User_1$ transfers a task to $User_2$, which is called social link. On account of there is a device connection relationship between the device of $User_1$ and the device of $User_2$, the task sent by $User_1$ can be offloaded to the device of $User_2$ for execution.

D. Problem Formulation

With the above system model, the problem of socially-aware dependent tasks offloading in MEC is mathematically formulated as follows. To better present the problem, the main symbols used throughout this paper are given in Table II.

TABLE II
THE MAIN SYMBOLS USED THROUGHOUT THIS PAPER

Symbols	Descriptions
ω_i^l	Energy consumption for local direct execution
ω_{ij}^d	Energy consumption for D2D-assisted local execution
ω_i^c	Energy consumption for offloading directly to the edge cloud
ω_{ij}^{dc}	Energy consumption for offloading to the edge cloud by D2D
ω	Energy consumption in general, *i.e.* $\omega = \{\omega_i^l, \omega_{ij}^d, \omega_i^c, \omega_{ij}^{dc}\}$
λ_i	Input data size of the task
ψ_i	Computing resources required to complete the task
μ_i	Output data size of the task
p_i^c	Energy consumption per CPU cycle for computation
H_{it}^d	D2D transmission power of device i
H_{ir}^d	D2D received power of device i
R_{ij}	D2D data rate between device i and j
H_{it}^c	Cellular transmission power
H_{ir}^c	Cellular Receiving Power
R_i^t	Upload cellular data rate
R_i^r	Download Cellular Data Rate
Z_v^m	Indicator of task v is offloaded to edge node m
t_v	Task v start time
$t_{v'}$	Task v' start time
m_v	Edge Nodes
T	Maximum completion time
U	Overhead function

Input: the inputs in a socially-aware dependent tasks offloading strategy based on dependencies in MEC include: tasks with dependencies, local mobile devices and edge cloud virtual machine devices, the property of tasks, the property of computing devices.

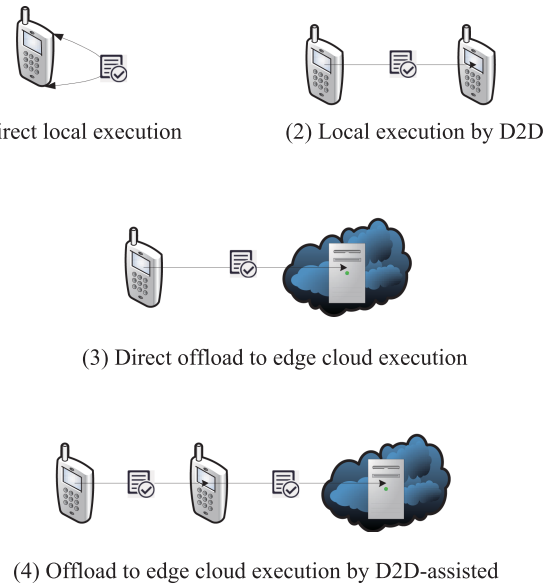
Constraints:

1) Constructing a DAG for dependent tasks;
2) A binary variable Z_v^m used to indicate whether task v is offloaded to edge node m with start time t_v and edge node m_v

- All tasks should be offloaded: $v \in V, \sum_{m\in M} Z_v^m = 1$;
- *Service constraint*: task $v \in V$ can only be offloaded to an edge node configured with the corresponding required service, such as edge node M_v, $\sum_{m\in M_v} Z_v^m = 1, \forall_v \in V$;
- *Dependency constraint*: for any $< v, v' > \in E$, all prior tasks are completed and the required data are sent to the edge node $m_{v'}$. Then, the task v' can be started execution, $< v, v' > \in E$, $t_v + \sum_{m\in M} Z_v^m t_{vm} + \sum_{m\in M}\sum_{m'\in M} c_{mm'} a_{vv'} Z_v^m Z_{v'}^{m'} \leq t_{v'}$;
- *Execute tasks sequentially*: If both v and v' are offloaded on the same edge node, each edge node may only perform one task at a time and the task cannot be stopped during execution, $t_v + t_{vm} = t_{v'}$;
- *Processing resource constraint*: each edge node has to satisfy the processing resource constraint, $\sum_{v\in V} Z_v^m t_{vm} r_{vm} \leq C(m), \forall_m \in M$.

The maximum completion time for these offloaded tasks is represented as

$$T = max\left\{t_v + \sum_{m\in M} Z_v^m t_{vm} + \sum_{m\in M}\sum_{m'\in M} c_{mm'} a_{vv'} Z_v^m Z_{v'}^{m'}, v \in V\right\} \quad (1)$$



(1) Direct local execution (2) Local execution by D2D

(3) Direct offload to edge cloud execution

(4) Offload to edge cloud execution by D2D-assisted

Fig. 5. Diagram of different offloading schemes.

The binary variable Z_v^m ($\forall_v \in V, m \in M$) is utilized to tackle the non-convex in (1) and further be rewritten as follows.

$$T = max\left\{t_v + \sum_{m\in M} Z_v^m t_{vm} + \sum_{m\in M}\sum_{m'\in M} c_{mm'}a_{vv'}max[Z_v^m + Z_{v'}^{m'} - 1, 0]\right\} \quad (2)$$

Since we focus on the current task offloading, and Z_v^m, $Z_{v'}^{m'}$ only takes 0 or 1. Thus, we simplify (2) as follows.

$$T = max\left\{t_v + \sum_{m\in M} Z_v^m t_{vm} + \sum_{m\in M} c_{mm'}a_{vv'}Z_v^m, v \in V\right\} \quad (3)$$

Output: an optimized offloading strategy with the minimal value of overhead function which takes both time and energy consumption into account with the above constraints.

IV. Offloading Strategy Selection

The device users have the flexibility to choose the optimal one from multiple task execution methods based on different users' requirements and device resource constraints. We can get the energy consumption for approaches of the direct local execution, the D2D-assisted local execution, the direct offload to edge cloud execution, and the offload to edge cloud execution by D2D-assisted, denoted as ω_i^l, ω_{ij}^d, ω_i^c, ω_{ij}^{dc}, respectively. A triple $< \lambda_i, \psi_i, \mu_i >$ is used to represent the input data size of the task, the computational resource required to complete the task, and the output data size of the task.

Fig. 5 depicts that four different offloading ways, including local direct execution, D2D-assisted local execution, direct offloading to edge cloud execution, and offloading to edge cloud execution via D2D.

A. Local Direct Execution

Device users can prefer to execute tasks locally in their mobile devices to avoid excessive overhead from offloading tasks. This is also can be viewed as a special case of task offloading.

Device i can execute its own tasks locally. Accordingly, the energy consumption is expressed,

$$\omega_i^l = p_i^c \psi_i \quad (4)$$

where p_i^c is used to calculate the energy cost per CPU cycle depending on various device types. ψ_i indicates the computing resources required for completing the task.

B. D2D-Assisted Local Execution

Two paired Devices can offload tasks with each other via D2D communication. In this way, they can share computational resources with each other efficiently.

The D2D connection allows device i to offload its own job to a neighbouring device j. Let H_{it}^d and H_{ir}^d indicate the D2D transmission power and received power, respectively, of device i, and R_{ij} denote the D2D transmission data rate between device i and device j. Thus, the energy consumption of the two devices for input and output data transmissions via the D2D transmission task is

$$Z_{ij}^{d1} = \frac{(H_{it}^d + H_{jr}^d)\lambda_i}{R_{ij}} + \frac{(H_{jt}^d + H_{ir}^d)\mu_i}{R_{ji}} \quad (5)$$

Besides, the energy consumption for performing the offloaded task on device j can be represented as

$$Z_{ij}^{d_2} = p_j^c \psi_i \quad (6)$$

Therefore, we can obtain the cost for D2D-assisted local execution as

$$\omega_{ij}^d = Z_{ij}^{d1} + Z_{ij}^{d2} \quad (7)$$

Since most mobile devices have the limited resource capacity, it is assumed that the device can only perform at the most one task at a time because of the physical size constraints.

C. Direct Offloading to Edge Cloud for Execution

The device can offload tasks to the edge cloud directly via the cellular link, thereby exerting its powerful cloud computing capabilities.

Through a cellular connection, device i may send its duties to an edge cloud server at the edge of network. The cellular transmission power and receiving power are H_{it}^c and H_{ir}^c, respectively. The upload and download cellular data rates are R_i^t and R_i^r, respectively. The energy consumption of the task input and output for transmitting data between the two devices via cellular communication. Due to the powerful computational capacity of the edge cloud, the energy consumption of the task execution is neglected in our work.

$$Z_i^c = \frac{H_{it}^c \lambda_i}{R_i^t} + \frac{H_{ir}^c \mu_i}{R_i^r} \quad (8)$$

Therefore, the cost for direct offloading to edge cloud for execution can be represented as follows,

$$\omega_i^c = Z_i^c \tag{9}$$

D. D2D-Assisted Offloading to Edge Cloud for Execution

A device with poor cellular connectivity can first send its computing tasks to a nearby device with high-quality cellular connectivity via a D2D connection. Then, this nearby device can help move tasks that require a lot of computing to the edge cloud.

The D2D link allows device i to communicate its own task data to the neighbouring device j. At the same time, it gets the result of the calculation from the adjacent device j. As a result, the energy consumption of the input and output data transfer through D2D communication between these two devices is as follows.

$$Z_{ij}^{d1} = \frac{(H_{it}^d + H_{jr}^d)\lambda_i}{R_{ij}} + \frac{(H_{jt}^d + H_{ir}^d)\mu_i}{R_{ji}} \tag{10}$$

The device j will then offload the received task from device i via the cellular link to the edge cloud server at the edge of network, and the energy consumption of the two devices for transferring the input task and output data transmission through D2D communication is as follows.

$$Z_j^c = \frac{H_{jt}^c \lambda_j}{R_j^t} + \frac{H_{jr}^c \mu_j}{R_j^r} \tag{11}$$

Consequently, the cost of the D2D-assisted offloading to edge cloud for execution is

$$\omega_{ij}^{dc} = Z_{ij}^{d1} + Z_j^c \tag{12}$$

E. Overhead Function

We define the following overhead function U by taking into account both time and energy consumption. Although execution latency and energy are metrics of two different dimensions, they can be combined into a hybrid metric by weighting them [19], [52].

$$U = \mu(T) + (1-\mu)(\omega) \tag{13}$$

where $\mu (0 \leq \mu \leq 1)$ denotes the weight of users' task execution time, and $(1-\mu)$ denotes the weight of energy consumption. Inspired by the existing work [53], we empirically divide the μ value into three cases:

1) if we pay more attention to execution time, then μ is set to 0.05;
2) if we pay more attention to energy consumption, then μ is set to 0.9;
3) if we balance execution time and energy consumption, then μ is set to 0.5.

F. Socially-Aware Dependent Tasks Offloading Based on Bipartite Graph Matching

Bipartite graph, a special model in graph theory, is utilized to model the allocation between tasks and devices. Let $G = (V, E)$ be an undirected graph, and a graph G is said to be a bipartite graph if the vertex V can be partitioned into two mutually disjoint subsets (A, B) and the two vertices i and j associated with each edge (i, j) in the graph belong to these two different sets of vertices $(i \in A, j \in B)$, respectively. In short, it means that the vertex set V can be partitioned into two mutually disjoint subsets, and the two vertices dependent on each edge in the graph belong to these two mutually disjoint subsets, and the vertices within the two subsets are not adjacent. Based on the definition of bipartite graph and the features of our problem, it is found that it is very convenient and effective to address the dependent task offloading by using the bipartite graph matching technique.

Our proposed strategy works as follows. First, the social graph of devices (Fig. 7(a)), the D2D connection graph (Fig. 7(b)), and the edge cloud with virtual machines VM 3 and VM 4 (Fig. 7(c)) are prepared, respectively. Then, a socially-aware collaborative task offloading can be formulated as bipartite graph matching problem (as shown in Fig. 6). And an illustration of the D2D-assisted edge computing offloading can be seen in Fig. 8. Clearly, it consists of physical layer and social layer. Mobile devices, in particular, contain a large number of D2D connections to ensure device computation and cellular communication resource sharing in the physical layer. Within the social layer, device users have social ties with one another in order to achieve effective and trustworthy collaboration.

By incorporating the social relationships and D2D communication relationships into the process of device computation and network resources sharing, this paper presents a socially-aware collaborative task offloading strategy based on bipartite graph matching. Once we obtain the energy consumption ω, then it can be regarded as a weight of each edge in the bipartite graph. Eventually, the optimal offloading strategy can be selected based on the weight.

In this paper, we assume that the number of executed tasks at the edge cloud is determined, then Fig. 6 can be simplified as Fig. 9(a).

When the task is not executed on the edge cloud server, there are two execution modes as shown in Fig. 9(a): direct local execution and D2D-assisted local execution, where the dotted line represents the connection which is obtained through social relationships. As shown in Fig. 9(a), Task 3 can be directly offloaded to Device 3, while Task 4 can be directly offloaded to Device 4. As can be seen from Fig. 7(a), there exists a social relationship between Device 1 and Task 3, so Task 3 can be executed locally by virtue of D2D communication with Device 1. Similarly, the Task 4 can be executed locally by virtue of D2D communication with Device 2.

When the task are offloaded to the edge cloud for execution, there exist two offloading strategies as shown in Fig. 9(b). That is to say, the tasks can be directly offloaded to the edge cloud for execution or offloaded to the edge cloud for execution via D2D communication. As can be seen from Fig. 9(b), task 3 can be directly offloaded to VM 3 and task 4 can be directly offloaded to VM 4 with the first offloading strategy. By utilizing the social relationships and D2D communication between devices, task 3 can be offloaded to VM 3 through device 1 and task 4 can be offloaded to VM 4 through device 2, that is the second offloading strategy.

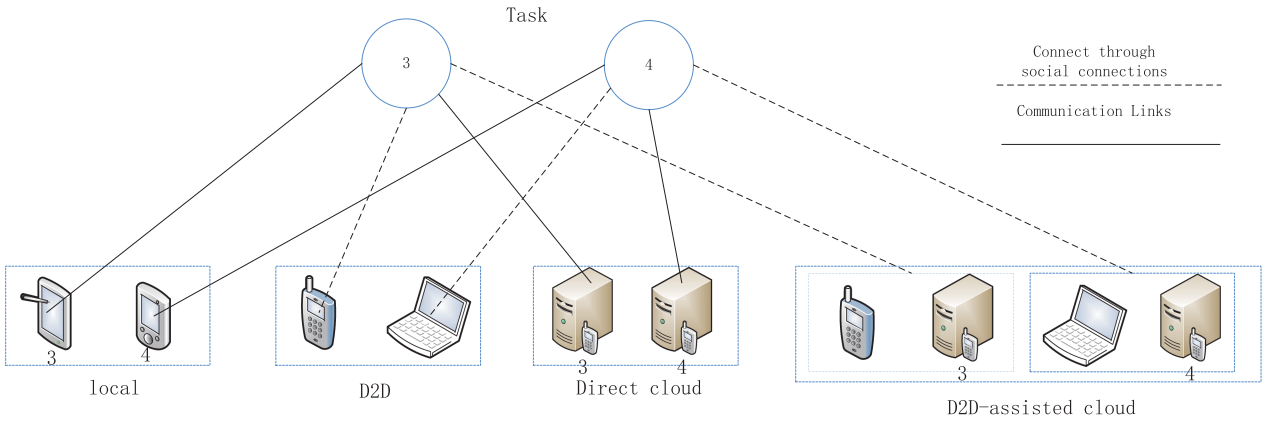


Fig. 6. Socially-aware collaborative task offloading based on bipartite graph matching.

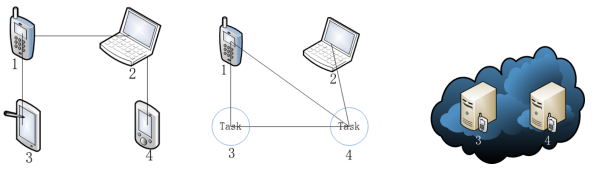


Fig. 7. Prerequisites for bipartite graph matching.

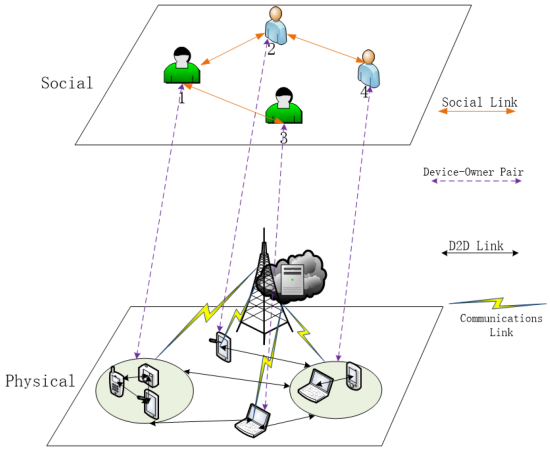
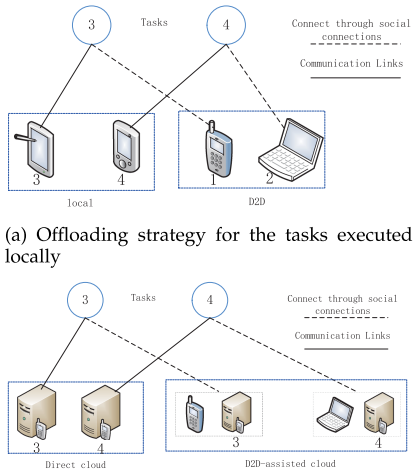


Fig. 8. An illustration of the D2D-assisted edge computing networks.



(a) Offloading strategy for the tasks executed locally

(b) Offloading strategy for the tasks executed in the edge cloud

Fig. 9. Offloading strategies for the tasks executed locally and in the edge.

First of all, we define relevant variables and characterization formulas used in our algorithm. After that, we impose time constraints on task offloading to satisfy the dependencies between tasks (Line 2). Then, Lines 3-6 calculate the energy consumption ω when executing task v locally; Lines 7-11 are in charge of evaluating the overhead U when executing task v locally. Similarly, when the task v is executed in the edge cloud, the energy consumption ω and the overhead U can be obtained (Lines 13-21); Then, the minimum value of U can be selected by comparing the overhead function U (Lines 22-27); Finally, an optimized combination of offloading strategies for tasks, is achieved according to the minimum value of U (Line 28). Note that, LOC indicates the tasks are execute locally; LOC-D2D refers to the tasks execution locally via D2D; EC denotes the tasks execution on the edge clouds and the EC-D2D represents the tasks execution on the edge clouds via D2D.

Theorem 4.1. Socially-aware Dependent Tasks Offloading (SDTO) algorithm is minimum weight matching under complete matching.

Proof. In a bipartite graph, there are two vertices: left vertex A and right vertex B. Now for each set of left connections A_i and right connections B_j have weights $U[i][j]$ to find the smallest match and minimize the sum of all $U[i][j]$ in the match. ■

Theorem 4.2. The task offloading strategy generated by the SDTO algorithm is the best for the current state.

Proof. In the problem proposed of this paper, all tasks are required to have corresponding task offloading strategies, and minimize the task latency and energy consumption. This means that the matches in the bipartite graph we build must be complete matches. The task offloading strategy obtained by SDTO

Algorithm 1: Socially-Aware Dependent Tasks Offloading Algorithm.

Input:
The property of tasks;
The property of computing devices
Output:
An optimized combination of offloading strategies for tasks E_n;

Define variables and characterization formula;
Characterize the dependence between tasks $t_v + \sum_{m\in M} Z_v^m t_{vm} + \sum_{m\in M}\sum_{m'\in M} c_{mm'} a_{vv'} Z_v^m Z_{v'}^{m'} \leq t_{v'}$
for (int $i = 0; i \leq n; i++$) **begin**
  Energy consumption $\omega = ((n-i) * execution\omega + i * offloading\omega)$;
  list.add(ω);
end
if (ω list ! $= null$) **begin**
for (int $i = 0; i < naList.size(); i++$) **begin**
  U: $u = offlo.computerU(T, naList.get(i))$;
  naListU.add(u);
  end
end
for (int $i = 0; i \leq n; i++$) **begin**
  $\omega = ((n-i)*$ the ω of EC $+i*$ the ω of EC-D2D);
  list.add(ω);
end
if (ω list != null) **begin**
for (int $i = 0; i < naList.size(); i++$) **begin**
  $u = offlo.computerU(T, naList.get(i))$;
  clListU.add(u);
end
begin
if(the U of LOC list naListU != null)**begin**
  Minimum U for LOC: minUNative $= Collections.min(naListU)$;
begin
if (the U of EC list clListU != null)**begin**
  Minimum U for EC: minU $= Collections.min(clListU)$;
Return E_n;

method according to Theorem 4.1 complete matches the minimum weight of the edge. Therefore, the task offloading strategy obtained by SDTO is the best decision in the current state. ■

Due to the maximum number of edges in a bipartite graph is $N^2/2$, the complexity of the Hungarian algorithm is classical $O(N^3)$ [54], hence, the complexity of the task offloading algorithm in this paper is $O(N^3)$ in each time interval. Notably, cubic complexity provides a theoretical upper bound on computational complexity. The total number of iterations depends on the number of vertices that need to be matched and the number of edges in the graph, that is to say, the number of tasks that need to be offloaded and the number of servers executing the tasks. In practice, the number of tasks that need to be offloaded and the number of mobile devices that provide assistance is usually limited in a time interval. In addition, bipartite graphs are not fully connected under normal conditions, therefore, we believe that the computational complexity of the algorithm proposed in this paper should be acceptable in practice.

G. An Illustrative Example

In order to better understand the algorithm proposed in this paper, an example will be described in this subsection. We assume that the set of tasks $V = \{v_1, v_2, v_3, v_4, v_5\}$ and the number of tasks $n = |V| = 5$, where there exists a dependency between tasks v_1 and v_2. In our simulation, there are 6 different offloading cases as follows. Case ① 5 tasks are executed locally; Case ② 5 task are executed on edge cloud; Case ③ 1 task is executed on edge cloud and 4 tasks are executed locally; Case ④ 2 tasks are executed on edge cloud and 3 tasks are executed locally; Case ⑤ 3 tasks are executed on edge cloud and 2 tasks are executed locally; Case ⑥ 4 tasks are executed on edge cloud and 1 task is executed locally.

① Fig. 10(a) shows the overhead of 6 different offloading strategies (i.e., $Loc(5), Loc - D2D(5), Loc(i) + Loc - D2D(j)$ (here $i + j = 5$)) under Case ①.[1]
Obviously, we can see that the offloading strategy $Loc(5)$ is the best one due to its smallest value of overhead U. Therefore, we can execute 5 tasks on the devices locally.

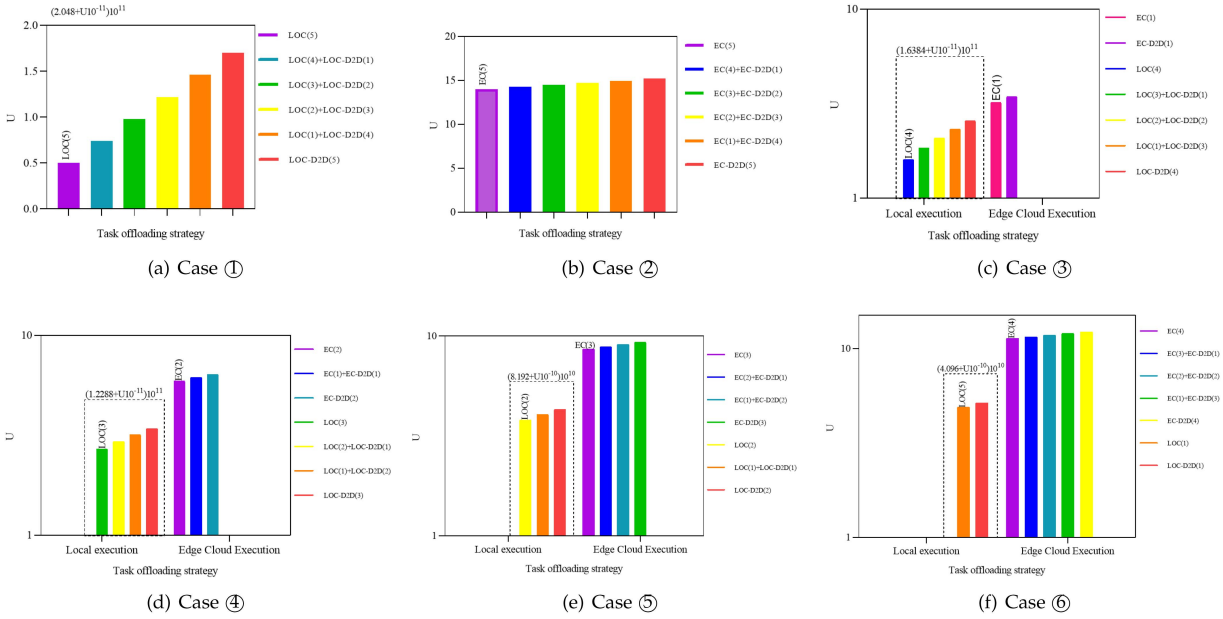
② Fig. 10(b) depicts the overhead of 6 different offloading strategies (i.e., $EC(5), EC - D2D(5), EC(i) + EC - D2D(j)$ (here $i + j = 5$)) under Case ②.
Similarly, it is easily to find that the offloading strategy $EC(5)$ is the best one due to its smallest value of overhead U. Therefore, we can offload 5 tasks to edge clouds for execution.

③ The large energy consumption of local execution in Fig. 10(c) results in a large difference between the values of various execution methods and further leads a tough observation and analysis for our results. To make Fig. 10(c) more intuitive for showing the results, the U value is taken as a logarithm operation, i.e, $log(U)$.
We find that the offloading strategy $EC(1)$+$Loc(4)$ is the best one due to its smallest value of overhead U. That is to say, we suggest offloading 1 task to edge cloud and 4 tasks to devices for execution.

④ Similar with Fig. 10(c), we also perform a logarithmic operation on the value of U on Fig. 10(d).
In Fig. 10(d), the offloading strategy $EC(2)$+$Loc(3)$ is the best one due to its smallest value of overhead U. Hence, we suggest offloading 2 tasks to edge cloud and 3 tasks to devices for execution.

⑤ For a better readability of our simulation results, we also perform a logarithmic operation on the value of U on Fig. 10(e). Fig. 10(e) illustrates that the offloading strategy $EC(3)$+$Loc(2)$ is the best one due to its smallest value of overhead U. Therefore, we suggest offloading 3 tasks to edge cloud and 2 tasks to devices for execution.

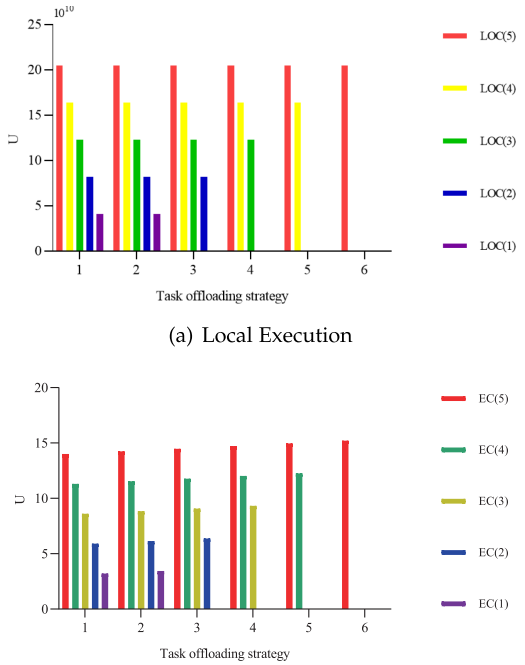
1.The numbers in parentheses indicate the number of tasks.



(a) Case ① (b) Case ② (c) Case ③

(d) Case ④ (e) Case ⑤ (f) Case ⑥

Fig. 10. Overhead of different offloading strategies under different cases.



(a) Local Execution

(b) Edge Cloud Execution

Fig. 11. The overhead U under different task offloading strategies.

⑥ Similar with Fig. 10(c), we also perform a logarithmic operation on the value of U on Fig. 10(f). Fig. 10(f) shows that the offloading strategy $EC(4)$+$Loc(1)$ is the best one due to its smallest value of overhead U. Therefore, we suggest offloading 4 tasks to edge cloud and 1 task to devices for execution.

Fig. 11 shows the summary of the different U-values for the tasks that are offloaded for execution on the local devices and edge cloud. It demonstrates that the U-values corresponding to the different offloading strategies (all 5 tasks are executed locally/on edge clouds, 4 out of 5 tasks are executed locally/on edge clouds, 3 out of 5 tasks are executed locally/on edge clouds, 2 out of 5 tasks are executed locally/on edge clouds, and 1 out of 5 tasks are executed locally/on edge clouds).

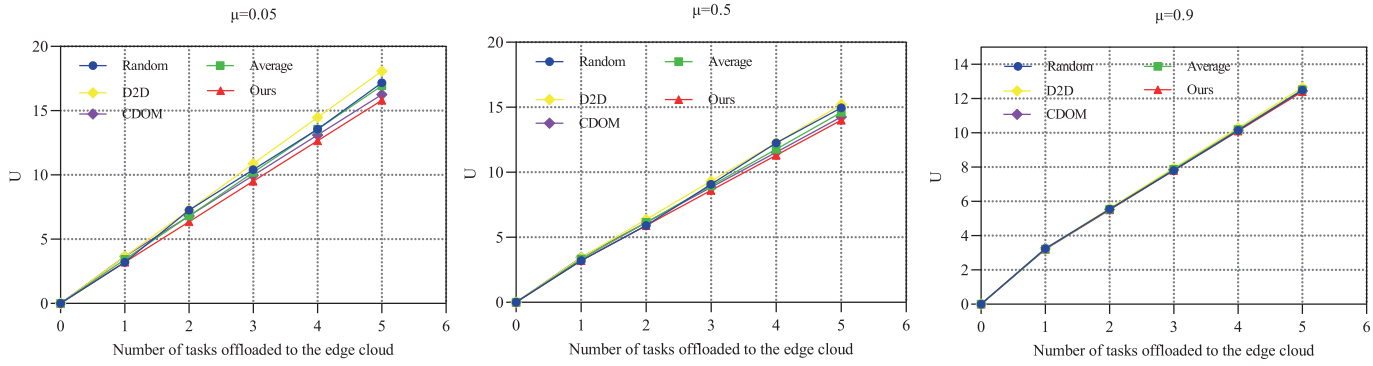
The summary of U-values executed locally and the summary of U-values executed on the edge cloud are complementary. For example, when 4 out of 5 tasks are executed locally, there is only 1 task executed on the edge cloud. Then, the U-value in this case is the sum of the corresponding U-values of the two graphs. It can be seen that with a total number of 5 tasks, the U-value can be minimized when all 5 tasks are offloaded to the edge cloud for execution. The corresponding offloading strategy is optimal.

V. Simulation and Performance Evaluation

In the simulation, we implement our proposed algorithm using Java programming language to perform all the simulations on a machine with Inter(R) Core (TM) i7-6700HQ @ 2.60 GHz 2.59 GHz CPU and 8 GB RAM running Windows 10 system. We verify the calculation of the overhead function U under a specific offloading case, and the simulation results demonstrate that our proposed strategy is more effective and sustainable.

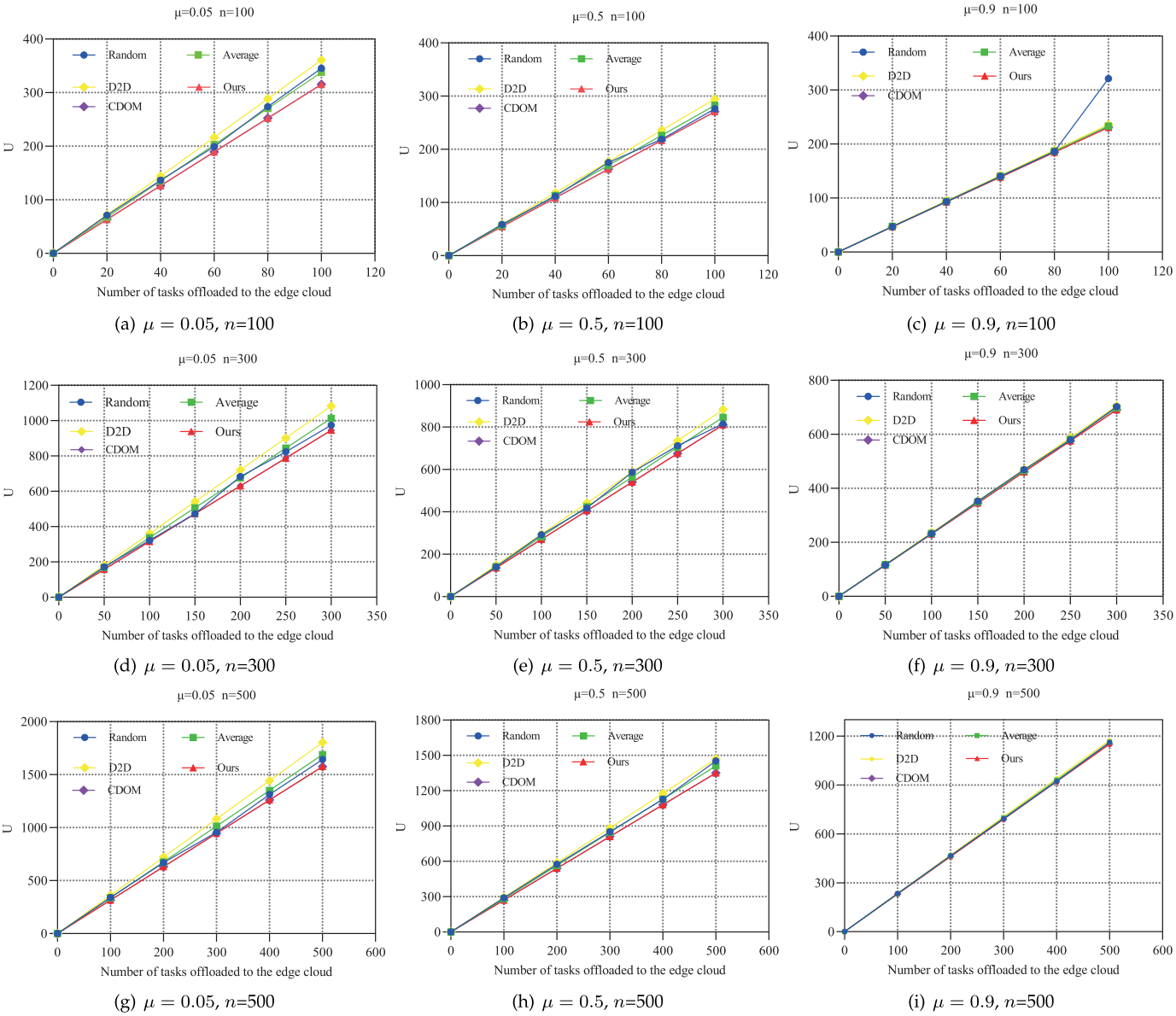
A. Experimental Settings

We assume that the set of tasks $V = \{v_1, v_2, \ldots, v_n\}$ and the number of tasks $n = |V| = \{100, 300, 500\}$. The ratio of communication delay to processing delay is uniformly randomized in the range of (0.1,10). In other words, the communication delay of data transfer from task v to task v' may be produced for each $< v, v' > \in E$ by multiplying task v's processing time by a random integer in (0.1,10). Furthermore, the necessary



(a) $\mu = 0.05$ (b) $\mu = 0.5$ (c) $\mu = 0.9$

Fig. 12. Robustness Evaluation of Our Approach when $n = 5$.



(a) $\mu = 0.05$, n=100 (b) $\mu = 0.5$, n=100 (c) $\mu = 0.9$, n=100

(d) $\mu = 0.05$, n=300 (e) $\mu = 0.5$, n=300 (f) $\mu = 0.9$, n=300

(g) $\mu = 0.05$, n=500 (h) $\mu = 0.5$, n=500 (i) $\mu = 0.9$, n=500

Fig. 13. Robustness evaluation of our approach varying μ and n.

TABLE III
PARAMETERS INITIALIZATION

Parameters	Value
λ_i	1.6×10^6bits
ψ_i	8192 bits
μ_i	0.5×10^3bits
p_i^c	1×10^7Hz
H_{it}^d	0.2W
H_{ir}^d	0.1W
H_{it}^c	2W
H_{ir}^c	1W
$R_{ij}/R_j^t/R_j^r$	10^6bps

computational resources t_{vm}, r_{vm} on edge node m for each task v are in the range of (1,10).

Suppose the start time $t_v = 1$, the execution time $t_{vm} = 2$, the number of CPU cycles per second $r_{vm} = 2$, and the data transmission delay is represented as follows,

$$a_{vv'}c_{mm'} = t_{vm} \times 0.1 = 2 \times 0.1 = 0.2 \tag{14}$$

The maximum completion time of the offloaded tasks can be estimated as follows,

$$T = max\left\{t_v + \sum_{m\in M} Z_v^m t_{vm} + \sum_{m\in M} c_{mm'}a_{vv'}Z_v^m, v \in V\right\}$$
$$= max\left\{1 + 2\sum_{m\in M} Z_v^m + 0.2\sum_{m\in M} Z_v^m, v \in V\right\} \tag{15}$$

We initialize the parameters used in our simulation as shown in Table III. In this paper, from the multi-objectives optimization point of view, we assume that all parameter values are not random. Therefore the difference in results of this paper is not due to the randomness of the data.

B. Simulation Results and Analysis

This section presents the experimental results of our simulation and relevant analysis.

To better evaluate the performance of our proposed offloading strategies, we utilize the following comparison approaches for further performance evaluation.

- *Random:* This method is to decide an offloading strategy according to the random selected value of U about different cases.
- *Average:* Similar with Random approach, this method decides an offloading strategy according to the average value of U of about different cases.
- *D2D:* The key idea of this approach is to select the way of D2D offloading as the task offloading strategy [38], including execution locally by virtue of D2D offloading, and execution on the edge clouds for D2D offloading.
- *CDOM:* The coordinate descent offloading method (CDOM) finds the optimal value through multiple iterations, and only changes the offloading strategy of one user each time in one iteration [30].
- *Ours:* The approach proposed in this paper. The core idea is to optimize the offloading strategy by considering the minimal value of U among different cases. Therefore, that is a global optimization solution for independent tasks offloading in MEC.

TABLE IV
THE RESULT OF THE FIG. 13(F)

Task number	Random	Average	CDOM	D2D	Ours
0	0	0	0	0	0
50	116.479	117.103	115.95	118.303	**115.9**
100	232.3	233.306	230.953	235.707	**230.905**
150	352.486	349.509	345.96	35.110	**345.908**
200	468.736	465.712	460.96	470.513	**460.91**
250	579.322	581.914	575.961	587.916	**575.913**
300	702.151	698.117	690.97	705.32	**690.915**

In order to evaluate the robustness of our approach, the following three groups of comparison experiments are conducted when $\mu = 0.05$, $\mu = 0.5$, and $\mu = 0.9$. That is to say, we set different weights μ for network latency and energy consumption, then have a comprehensive observation on performance of comparison approaches.

Fig. 12(a), (b), and (c) simulate the variation of the overhead for different approaches obtained when the total number of tasks is 5, as the number of tasks offloaded to the edge cloud increases, and is compared at different μ values. The results show that the value of the overhead obtained by our approach is smaller than the other four approaches, in which our approach still outperforms the others for different values of μ.

Since it is difficult to generalize the results obtained with a small number of tasks, we adjusted the total number of tasks to address this issue. Without loss of generality, we set the total number of tasks to 100, 300, and 500 under the same μ value respectively, in order to make the obtained results more convincing.

Fig. 13(a), (b), and (c) demonstrate that the variation of the different overhead functions corresponding to the number of tasks offloaded on the edge cloud for different offloading when the total number of tasks is 100. It can be seen that the comparison of overhead functions at different μ values is always better for our method than the other four approaches.

Robustness Evaluation: Fig. 13(d)–(i) show the robustness evaluation of our approach. It reports that our approach can achieve the minimal value of U compared with other approaches under different weights μ and the number of tasks 300 and 500.

Note that, as shown in Figs. 12(c), 13(f), and 13(i), although the difference among them is not significant, take the Table IV as an example we can still conclude that our approach is superior to the others in terms of U. Because the value of μ at this time is taken as 0.9, which means that the value of the influence of T on the overhead function accounts for 90% at this time. According to (3), the factor affecting T is decided by the number of tasks offloaded to the edge cloud. As a result, the difference in overhead function values achieved for a given total number of tasks versus the number of jobs offloaded to the edge cloud during simulation tests is rather modest.

Sensitivity Analysis: We change the parameter values that can be randomly set in this paper, and conduct 100 times of

TABLE V
CONFIDENCE INTERVAL ESTIMATION

Number of tasks	μ=0.05		μ=0.5		μ=0.9	
	Confidence Interval	$\overline{U}$	Confidence Interval	$\overline{U}$	Confidence Interval	$\overline{U}$
5	(33.47,33.71)	33.59	(31.26,33.80)	32.53	(31.65,35.82)	33.73
100	(667.45,672.18)	669.81	(634.54,687.72)	661.13	(602.98,686.23)	644.60
300	(2227.37,2243.15)	2235.26	(2066.71,2231.29)	2149	(2043.02,2314.58)	2178.80
500	(3341.49,3365.41)	3353.45	(3107.50,3344.11)	3225.80	(2927.54,3341.74)	3134.64

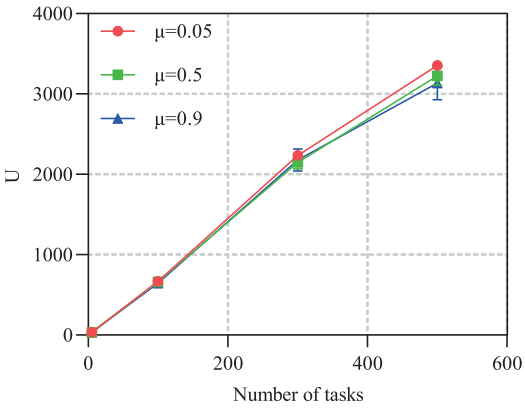


Fig. 14. Sensitivity analysis results.

experiments under 12 different settings with various μ values and task number. Furthermore, we estimate the 95% confidence interval of the experimental results, i.e.,$(\overline{U} - \frac{1.96*\sigma}{\sqrt{L}}, \overline{U} + \frac{1.96*\sigma}{\sqrt{L}})$, where σ and $\overline{U}$ denote the Standard Deviation and mean value of overhead U and the number of experiments L=100. Table V and Fig. 14 present the confidence interval and sensitivity analysis results. On the premise of ensuring the confidence level of 95%, the lengths of the confidence interval of the overhead U under 12 different experimental configurations are quite short, which is conducive to making the appropriate dependency tasks offloading decision, which fully proves that the experimental results obtained by our method are not due to the randomness of the data.

VI. CONCLUSION

In this paper, we exploit the social relationships between mobile devices to enable trusted transmission between dependent tasks and other tasks. Our proposed socially-aware dependent task offloading strategies in mobile edge computing provide more flexible options for task offloading, including local execution on the devices, execution locally by virtue of D2D offloading, execution on the edge clouds directly and execution on the edge clouds by means of D2D offloading, for achieving the efficient offloading of dependent tasks. Technically, a bipartite graph matching based offloading approach for dependent tasks is presented. We consider both network latency and energy consumption. Therefore, the final adopted offloading strategy demonstrates lower overhead and computational cost, as well as better quality of experience than considering only one of the factors. Extensive simulation results demonstrate that the offloading of dependent tasks with our approach is less expensive than the approaches considering time or energy consumption only. In future work, we will study the problem of offloading dependent tasks under heterogeneous computing resources using graph neural methods. Moreover, we try to do some experiments in real scenarios as much as possible.

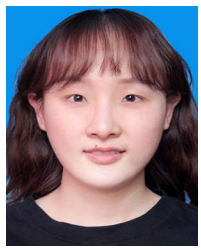
ACKNOWLEDGMENTS

This work reflects only the authors' view and the EU Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

[1] N. Panwar, S. Sharma, and A. K. Singh, "A survey on 5G: The next generation of mobile communication," *Phys. Commun.*, vol. 18, pp. 64–84, 2016.
[2] J. G. Andrews et al., "What will 5G be?," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 6, pp. 1065–1082, Jun. 2014.
[3] R. Anwit, P. K. Jana, and A. Tomar, "Sustainable and optimized data collection via mobile edge computing for disjoint wireless sensor networks," *IEEE Trans. Sustain. Comput.*, vol. 7, no. 2, pp. 471–484, Second Quarter 2022.
[4] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1468–1476.
[5] 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.
[6] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
[7] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surv. Tut.*, vol. 19, no. 3, pp. 1628–1656, Third Quarter 2017.
[8] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.
[9] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.
[10] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2017, pp. 1–6.
[11] L. Tong, Y. Li, and W. Gao, "A hierarchical edge cloud architecture for mobile computing," in *Proc. IEEE 35th Annu. Int. Conf. Comput. Commun.*, 2016, pp. 1–9.
[12] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 37–45.
[13] J. Zhu, X. Li, R. Ruiz, and X. Xu, "Scheduling stochastic multi-stage jobs to elastic hybrid cloud resources," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1401–1415, Jun. 2018.
[14] J. Ni, A. Zhang, X. Lin, and X. S. Shen, "Security, privacy, and fairness in fog-based vehicular crowdsensing," *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 146–152, Jun. 2017.
[15] E. D'Andrea, P. Ducange, B. Lazzerini, and F. Marcelloni, "Real-time detection of traffic from Twitter stream analysis," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 4, pp. 2269–2283, Aug. 2015.
[16] E. Cuervo et al., "MAUI: Making smartphones last longer with code offload," in *Proc. 8th Int. Conf. Mobile Syst. Appl. Serv.*, 2010, pp. 49–62.

[17] X. Chen, Z. Zhou, W. Wu, D. Wu, and J. Zhang, "Socially-motivated cooperative mobile edge computing," *IEEE Netw.*, vol. 32, no. 6, pp. 177–183, Nov./Dec. 2018.
[18] X. Pei, W. Duan, M. Wen, Y.-C. Wu, H. Yu, and V. Monteiro, "Socially-aware joint resource allocation and computation offloading in noma-aided energy harvesting massive IoT," *IEEE Internet Things J.*, vol. 8, no. 7, pp. 5240–5249, Apr. 2021.
[19] 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.
[20] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.
[21] L. Ma, X. Wang, X. Wang, L. Wang, Y. Shi, and M. Huang, "TCDA: Truthful combinatorial double auctions for mobile edge computing in industrial Internet of Things," *IEEE Trans. Mobile Comput.*, vol. 21, no. 11, pp. 4125–4138, Nov. 2021.
[22] J. Bright, "The success of multi-access edge computing rests on cross-industry cooperation and 5G," *3G Mobile*, vol. 13, pp. 2–5, 2017.
[23] L. Zhao, K. Yang, Z. Tan, X. Li, S. Sharma, and Z. Liu, "A novel cost optimization strategy for SDN-enabled UAV-assisted vehicular computation offloading," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 6, pp. 3664–3674, Jun. 2021.
[24] F. Liu, P. Shu, and J. C. Lui, "AppATP: An energy conserving adaptive mobile-cloud transmission protocol," *IEEE Trans. Comput.*, vol. 64, no. 11, pp. 3051–3063, Nov. 2015.
[25] P. Jin, X. Fei, Q. Zhang, F. Liu, and B. Li, "Latency-aware VNF chain deployment with efficient resource reuse at network edge," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 267–276.
[26] A. Yadav, P. K. Jana, S. Tiwari, and A. Gaur, "Clustering-based energy efficient task offloading for sustainable fog computing," *IEEE Trans. Sustain. Comput.*, early access, Jun. 27, 2022, doi: 10.1109/TSUSC.2022.3186585.
[27] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.
[28] X. Chen, L. Pu, L. Gao, W. Wu, and D. Wu, "Exploiting massive D2D collaboration for energy-efficient mobile edge computing," *IEEE Wireless Commun.*, vol. 24, no. 4, pp. 64–71, Aug. 2017.
[29] X. Yi, L. Pan, Y. Jin, F. Liu, and M. Chen, "Edirect: Energy-efficient D2D-assisted relaying framework for cellular signaling reduction," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 860–873, Apr. 2020.
[30] S. Yang, "A joint optimization scheme for task offloading and resource allocation based on edge computing in 5G communication networks," *Comput. Commun.*, vol. 160, pp. 759–768, 2020.
[31] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang, "Joint computation offloading and user association in multi-task mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 12, pp. 12 313–12 325, Dec. 2018.
[32] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Trans. Serv. Comput.*, vol. 12, no. 5, pp. 726–738, Sep./Oct. 2019.
[33] Z. Xu, W. Liang, M. Jia, M. Huang, and G. Mao, "Task offloading with network function requirements in a mobile edge-cloud network," *IEEE Trans. Mobile Comput.*, vol. 18, no. 11, pp. 2672–2685, Nov. 2018.
[34] X. Meng, W. Wang, and Z. Zhang, "Delay-constrained hybrid computation offloading with cloud and fog computing," *IEEE Access*, vol. 5, pp. 21 355–21 367, 2017.
[35] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.
[36] L. Zhao, T. Zheng, M. Lin, A. Hawbani, and J. Shang, "A novel influence maximization-based dynamic forwarding node selection scheme in SD-VNs," in *Proc. IEEE Int. Conf. Parallel Distrib. Process. Appl. Big Data Cloud Comput. Sustain. Comput. Commun. Social Comput. Netw.*, 2020, pp. 674–682.
[37] S. Yu, B. Dab, Z. Movahedi, R. Langar, and L. Wang, "A socially-aware hybrid computation offloading framework for multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 6, pp. 1247–1259, Jun. 2020.
[38] N. Fan, X. Wang, D. Wang, Y. Lan, and J. Hou, "A collaborative task offloading scheme in D2D-assisted fog computing networks," in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2020, pp. 1–6.
[39] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1997–2006.
[40] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 207–215.
[41] V. Farhadi et al., "Service placement and request scheduling for data-intensive applications in edge clouds," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1279–1287.
[42] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 514–522.
[43] J. Lin, R. Chai, M. Chen, and Q. Chen, "Task execution cost minimization-based joint computation offloading and resource allocation for cellular D2D systems," in *Proc. IEEE 29th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun.*, 2018, pp. 1–5.
[44] 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. 11 255–11 268, 2017.
[45] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 883–893, Jun. 2020.
[46] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2020.
[47] 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.
[48] J. Zou, T. Hao, C. Yu, and H. Jin, "A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 228–239, Feb. 2020.
[49] Y. Shang, J. Li, and X. Wu, "Dag-based task scheduling in mobile edge computing," in *Proc. 7th Int. Conf. Inf. Sci. Control Eng.*, 2020, pp. 426–431.
[50] J. Chen, Y. Yang, C. Wang, H. Zhang, C. Qiu, and X. Wang, "Multi-task offloading strategy optimization based on directed acyclic graphs for edge computing," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9367–9378, Jun. 2022.
[51] X. Lv, H. Du, and Q. Ye, "TBTOA: A dag-based task offloading scheme for mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, 2022, pp. 4607–4612.
[52] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, Apr. 2018.
[53] Q. Tang, H. Lyu, G. Han, J. Wang, and K. Wang, "Partial offloading strategy for mobile edge computing considering mixed overhead of time and energy," *Neural Comput. Appl.*, vol. 32, no. 19, pp. 15 383–15 397, 2020.
[54] J. Xie, Y. Jia, Z. Chen, Z. Nan, and L. Liang, "D2D computation offloading optimization for precedence-constrained tasks in information-centric IoT," *IEEE Access*, vol. 7, pp. 94 888–94 898, 2019.



Yanqi Gong received the BSc degree in software engineering from Ningxia University, China, in 2020. She is currently working toward the MSc degree with the School of Computer Science, Shaanxi Normal University, China. Her current research interests include mobile edge computing, cloud computing, and Big Data.

Fei Hao received the PhD degree in computer science and engineering from Soonchunhyang University, South Korea, in 2016. Since 2016, he has been with Shaanxi Normal University, Xi'an, China, where he is an associate professor. From 2020 to 2022, he was a Marie Sklodowska-Curie Fellow with the University of Exeter, Exeter, United Kingdom. His research interests include social computing, ubiquitous computing, big data analytics, knowledge graph, and edge intelligence.

Liang Wang (Member, IEEE) received the BS degree in telecommunications engineering and the PhD degree in communication and information systems from Xidian University, China, in 2009 and 2015, respectively. He is currently an associate professor with the School of Computer Science, Shaanxi Normal University, China. From 2018 to 2019, he was a Visiting Scholar with the School of Electrical and Computer Engineering, Georgia Institute of Technology, USA. His research interests focus on Internet of Things, mobile edge computing, and applications of reinforcement learning and robust design in wireless communications networks.

Liang Zhao (Member, IEEE) received the PhD degree from the School of Computing, Edinburgh Napier University, in 2011. He is a professor with Shenyang Aerospace University, China. Before joining Shenyang Aerospace University, he worked as associate senior Researcher in Hitachi (China) Research and Development Corporation from 2012 to 2014. He is also a JSPS invitational Fellow (2023). He was listed as Top 2% of scientists in the world by Standford University (2022). His research interests include ITS, VANET, WMN and SDN. He has published more than 150 articles. He served as the Chair of several international conferences and workshops, including 2022 IEEE BigDataSE (Steering Co-Chair), 2021 IEEE TrustCom (Program Co-Chair), 2019 IEEE IUCC (Program Co-Chair), and 2018-2022 NGDN workshop (founder). He is associate editor of Frontiers in Communications and Networking and Journal of Circuits Systems and Computers. He is/has been a guest editor of *IEEE Transactions on Network Science and Engineering*, *Springer Journal of Computing*, etc. He was the recipient of the best/outstanding paper awards, 2015 IEEE IUCC, 2020 IEEE ISPA, 2022 IEEE EUC, and 2013 ACM MoMM.

Geyong Min (Member, IEEE) received the PhD degree in computing science from the University of Glasgow, United Kingdom, in 2003, and the BSc degree in computer science from Huazhong University of Science and Technology, China, in 1995. He is a professor of High Performance Computing and Networking in the Department of Computer Science within the College of Engineering, Mathematics and Physical Sciences with the University of Exeter, United Kingdom. His research interests include Computer Networks, Wireless Communications, Parallel and Distributed Computing, Ubiquitous Computing, Multimedia Systems, Modelling, and Performance Engineering.